НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

кафедра автоматика та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК 004.93

«До захисту допущено»

Завідувач кафедри
_____ О. І. Ролік
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 126 «Інформаційні системи та технології»
(код і назва спеціальності)

на тему: Методологія імплементації навчання з підкріпленням у відеоігровому штучному інтелекті

Виконав : студент 6 курсу, групи ІА–71мн
(шифр групи)

Шпита Гліб Володимирович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н. Дорогий Я.Ю. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент доцент кафедри ІБ, к.т.н. доц. Демчинський В.В. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

РЕФЕРАТ

В даній роботі 102 сторінки текстової інформації, 24 рисунки, 28 таблиць, 9 додатків.

Актуальність даної роботи полягає в наявності проблеми використання громіздких негнучких статичних алгоритмів прийняття рішень при розробці штучного інтелекту для відеоігор, що призводить до високої складності перепрограмування ігрових агентів у випадку введення змін у ігровій механіці. Засоби машинного навчання можуть вирішити дану проблему, але вони рідко застосовуються у сфері. Така тенденція пов'язана з декількома факторами, основним з яких є відсутність даних для тренування алгоритму, що ускладнює категоризацію рішень агента як «правильні» та «неправильні». До того ж часто існує можливість багаторазової зміни правил та функцій ігрового середовища у ході розробки, що призведе до необхідності постійного перенавчання алгоритму або змін у його архітектурі, що згодом може призвести до несподіваної та неадекватної поведінки агенту через додавання до середовища нових параметрів, необхідність в чому часто виникає вже після релізу гри з метою балансування, або випуску нового контенту.

Метою даної магістерської дисертації є створення методології застосування алгоритмів машинного навчання з підкріпленням у розробці відеоігор для отримання агентів, що здатні пристосовуватися до змін у ігровому середовищі. Розроблена методологія може бути використана розробниками у даній сфері для ефективного рішення проблеми прийняття рішень агентами за умов змінних параметрів ігрового середовища без втрати можливості контролю їх пріоритетів та налаштування бажаних шаблонів поведінки.

Об'єктом є відеоігрові штучні інтелекти. Предметом є алгоритми навчання з підкріпленням, що використовуються для пошуку оптимальної поведінки за встановленим критерієм.

Ключові слова: відеоігри, штучний інтелект, навчання з підкріпленням, адаптивні алгоритми.

The relevance of this work is the presence of the problem of using cumbersome inflexible static decision-making algorithms in field of development of artificial intelligence for video games, which leads to a high complexity of reprogramming gaming agents in case of introducing changes in game mechanics. Machine learning tools can solve this problem, but they rarelu used in the field. This tendency exists due to several factors, the main of which is the lack of data for algorithm training, which complicates the categorization of the decisions of the agent as "correct" and "wrong". In addition, there is often exists the possibility of multiple changes to the rules and functions of the game environment during development, which will necessitate constant overhaul of the algorithm or changes in its architecture, which can subsequently lead to unexpected and inadequate behavior of the agent after adding new parameters to the environment, the need of which often occurs after the game's release to balance or release new content.

The purpose of this master's dissertation is to create a methodology for the application of reinforcement learning algorithms in the development of video games to obtain agents that are able to adapt to changes in the gaming environment. The developed methodology can be used by developers in this field to effectively solve the problem of decision-making by agents in the conditions of changing parameters of the gaming environment without losing the ability to control their priorities and to configure desired behavior patterns.

The object is videogame artificial intelligence. The subject is algorithms of reinforcement learning, which are used to find optimal behavior according to the established criterion.

Keywords: video games, artificial intelligence, training with reinforcement, adaptive algorithms.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – Штучний інтелект (У контексті даної дисертації мається на увазі ігровий штучний інтелект).

RL – Reinforcement Learning (Навчання з підкріпленням).

MDP – Markov Decision Process (Марковський процес прийняття рішень).

DQN – Deep Q-Network. Технологія Q-навчання зі застосуванням глибокого навчання.

DDQN – Double Deep Q-Network. Технологія Q-навчання зі застосуванням глибокого навчання і комбінації двох політик прийняття рішень.

GAN – Generative adversarial network (Генеративно-змагальна мережа).

Rogue – Жанр відеоігор, особливістю якого є випадково створювані рівні і незворотність смерті персонажа.

NPC – Non-Player Character. Персонаж у грі, що не контролюється гравцем.

Перк – Особливість персонажу гри, яка впливає на ігровий процес.

Atari – Французька компанія по виробництву ігрових консолей та ігор. У контексті даної дисертації даний термін означає консоль даної компанії – Atari 2600.

Агент – Програмний модуль що існує у програмному середовищі і виконує певне завдання протягом тривалого проміжку часу.

RL-Агент – Агент, що використовує навчання з підкріпленням.

ВСТУП

Алгоритми машинного навчання набувають все більшої популярності у різноманітних областях, ефективно вирішуючи задачі розпізнавання, класифікації, прогнозування та прийняття рішень. Нейронні мережі застосовуються для написання музики, створення картин, вирішення комплексних задач регулювання параметрів різноманітних систем. Тим не менш, все ще існують області, де машинне навчання майже не використовується і перевага надається звичайним громіздким негнучким статичним алгоритмам як, наприклад, дерева рішень, що проектуються безпосередньо розробниками програмного забезпечення цілком на основі їх досвіду Однією з найбільших таких областей є розробка відеоігор, а конкретніше – розробка відеоігрового штучного інтелекту. У даній сфері штучним інтелектом називають системи, що відповідають за управління ігровим середовищем та агентами, з якими взаємодіє гравець. Створюючи ігровий штучний інтелект, розробники, зазвичай використовують так звані «машини станів», у яких конкретно визначають певний набір ігрових станів відносно точки зору комп'ютерного агенту на основі набору змінних параметрів ігрового середовища, на які може впливати гравець, таким чином, взаємодіючи з агентами. При програмуванні машин станів, розробники визначають які саме завчасно передбачені дії може або має виконати агент, будучи у певному стані, таким чином, застосовуючи даний підхід, програміст ігрового штучного інтелекту зобов'язаний передбачити якнайбільшу кількість можливих ігрових сценаріїв на основі власного досвіду у проектування аналогічних систем, що буває довготривалим громіздким процесом. До того ж, програмування поведінки штучного інтелекту на основі суб'єктивних рішень окремо для різних ігрових станів може призвести до нелогічної поведінки агентів у випадку потрапляння у стан, що не був передбачений розробником, що часто трапляється у даній сфері[1].

На перший погляд, застосування технік машинного навчання здається очевидним рішенням вказаних проблем, до того ж, у сучасній ігровій індустрії зацікавлені сотні мільйонів людей, тому, безумовно, питання винайдення нових

ігрових механік є надзвичайно критичним питанням для розробників, які бажають привернути увагу аудиторії. Не дивлячись на це, у відеоігровій сфері продовжується тотальне переважання систем на основі дерев рішень[2]. Така тенденція пов’язана з декількома факторами, основним з яких є відсутність даних для тренування алгоритму, що ускладнює категоризацію рішень ігрового агента як «правильні» та «неправильні», що, в свою чергу, робить неможливим застосування систем на основі машинного навчання з вчителем. Більшість ігрових компаній підходять до цього питання консервативно, не схиляючись до експериментуванням з рідкісними ігровими механіками через можливу економічну недоцільність такого рішення та перевірених часом класичних алгоритмів. Також, значною проблемою є відсутність повного контролю над біхевіоральними параметрами ШІ на основі машинного навчання, оскільки подібні системи, зазвичай, є «чорним ящиком», який налаштовується автоматично по мірі тренування і не допускає можливості вільного редагування розробником.

Метою даної магістерської дисертації є створення методології використання засобів машинного навчання у відеоігрових системах з метою покращення якості поведінки комп’ютерних агентів у непередбачуваних ситуаціях та створення більш гнучких та незалежних від оновлень гри систем штучного інтелекту. Таким чином, розроблюваний за даною методологією ігровий штучний інтелект має не лише бути спроможним шукати оптимальні за правилами гри рішення без валідації вчителем, але й бути стійким до непередбачуваних змін в умовах середовища. До того ж, структура даного штучного інтелекту має дозволяти виконувати модифікації у поведінці агента в залежності від потреб розробника. Дана проблема представляє собою пласт задач з імплементації адаптивних алгоритмів у динамічних системах без даних для навчання.

Основними задачами є:

- Аналіз існуючих рішень у області застосування машинного навчання у відеоігрових системах штучного інтелекту як академічних, так і практичних;
- Визначити переваги та недоліки кожного з запропонованих методів, провести аналіз їх ефективності, складності реалізації;

- Змоделювати тестове ігрове середовище з можливістю динамічних змін параметрів;
- У розробленому середовищі виконати експеримент у вигляді введення ігрових агентів зі застосуванням наведеної методології та порівняти ефективність роботи агентів різних класів;
- На основі отриманих даних розробити набір методів використання машинного навчання у ШІ в залежності від особливостей гри;
- Представити результати у вигляді стартап проекту.

Об’єктом дослідження магістерської дисертації є системи відеоігрових штучних інтелектів.

Предметом дослідження є алгоритми машинного навчання, що використовуються для пошуку оптимальної поведінки агента за встановленим критерієм.

Основним здобутком дисертації є розроблена методологія використання механізмів машинного навчання у відеоігрових ШІ, яка може бути використана розробниками у даній сфері для ефективного рішення проблеми прийняття рішень агентами за умов змінних параметрів ігрового середовища без втрати можливості контролю їх пріоритетів та налаштування бажаних шаблонів поведінки. Центральним методом реалізації адаптивності у наведеній методології є застосування навчання з підкріпленням, яке, згідно до результатів тестів, демонструє збільшення ефективності рішень агентів на 75% при застосуванні правил розробленої методології в умовах динамічного ігрового середовища у порівнянні зі агентами на основі класичних дерев рішень. Використання наведених методів дозволить розробникам ігрових програм створювати ефективні системи прийняття рішень без попереднього тренування або побудови комплексних дерев рішень як для задання поведінки ігровим агентам, так і для адаптивного керування глобальними ігровими параметрами.

1 ПОНЯТТЯ ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ

На найпростішому рівні «ігровий штучний інтелект» полягає в моделюванні або імітації поведінки персонажів гри або об'єктів (тобто всіх елементів гри з якими може взаємодіяти гравець). У традиційних дослідженнях в галузі ШІ метою є створення справжнього інтелекту, або навіть штучного розуму штучними засобами. З точки зору ігор справжній ШІ надзвичайно виходить за рамки вимог розважального програмного проекту. В іграх така потужність, зазвичай, не потрібна. Ігровий ШІ не повинен бути наділений почуттями і самосвідомістю. Йому немає необхідності навчатися чогось за межами рамок ігрового процесу. Мета ШІ в іграх полягає в імітації розумної поведінки персонажів та управління параметрами середовища[3] і до ШІ часто відносять попередньо запрограмовані сценарії ігрових подій, які називаються «скриптами».

Залежно від характеру і ролі ШІ в грі вимоги до ресурсів можуть бути доволі незначними. На базовому рівні необхідні лише затрати часу роботи процесора. У більш складних випадках потрібні засоби аналізу середи ШІ, реєстрації дій гравця і оцінки успішності попередніх дій. Основним принципом, що лежить в основі роботи штучного інтелекту, є прийняття рішень. Для вибору варіанту дії система повинна впливати на об'єкти за допомогою системи ШІ. При цьому такий вплив може бути організовано у вигляді «мовлення ШІ» або «звернень об'єктів»[4].

У системах з «мовленням ШІ» система прийняття рішень зазвичай ізольована у вигляді окремого елемента ігровий архітектури. Така стратегія часто приймає форму окремого потоку або декількох потоків, в яких ШІ обчислює найкраще рішення для заданих параметрів гри. Коли ШІ приймає рішення, воно передається усім об’єктам, що беруть участь. Такий підхід найкраще працює в стратегіях реального часу, де ШІ аналізує загальний хід подій у всій грі[5].

Системи зі «зверненнями об'єктів» краще підходять для ігор з простими елементами. В таких іграх об'єкти звертаються до системи ШІ кожен раз, коли об'єкт повинен виконати дію або оновити свій стан. Такий підхід є ефективним для систем з великою кількістю об'єктів, яким не потрібно приймати нові рішення занадто часто,

наприклад в шутерах та інших іграх, де поведінку NPC можливо описати простими правилами. Така система також може скористатися перевагами багатопотокової архітектури, але для неї потрібно більш складне планування.

Існує два основних способи програмування ШІ, які активно використовуються і відеоігровій сфері: дерева рішень та машини станів. Дерево рішень - алгоритм для прийняття рішень, який уявляє собою структуру, подібну до блок-схеми, в якій кожен внутрішній вузол являє собою правило щодо значення певного атрибуту. Кожна гілка являє собою один з основних можливих варіантів станів, а кожен вузол уявляє собою дію, яку система має виконати у випадку, виконання усіх правил, що привели до даного вузла. Таким чином, шляхи від кореня до листа являють собою правила класифікації ігрових станів, встановлені розробником на основі бажаного шаблону поведінки ігрових агентів. У випадку використання дерева рішень, ШІ на кожній ітерації ігрового середовища проходить по запрограмованому набору правил, перевіряючи вірність вказаних умов і виконуючи дію, яка відповідає поточному стану. Машина станів – це абстрактний автомат з деякою кінцевою кількістю станів, перехід між якими можливий у випадку виконання деяких встановлених правил. Зазвичай, перебування агента в певному стані супроводжується деякою запрограмованою поведінкою, які продовжує виконуватися, доки не відбудеться певна подія, яка може змінити стан автомату на інший. Стани, поведінка у них та події-тригери зміни стану, зазвичай, визначаються розробниками відповідно до розроблюваної ігрової механіки.

Щоб штучний інтелект міг приймати логічні осмислені рішення, йому необхідно певним чином сприймати середу, в якій він знаходиться, інакше кажучи: експлуатувати зворотний зв'язок зі своїм середовищем. У простих системах подібна перцепція може обмежуватися простою перевіркою положення у просторі об'єкта, яким управляє гравець. У більш складних системах потрібно визначати основні характеристики і властивості ігрового світу, наприклад можливі маршрути для пересування, наявність природних укриттів на місцевості, області конфліктів. При цьому розробники мають продумувати спосіб виявлення і визначення основних властивостей ігрового світу, важливих для системи ШІ. Наприклад, укриття на

місцевості можуть бути заздалегідь визначені дизайнерами рівнів або заздалегідь обчислені при завантаженні або компіляції карти рівня. Деякі елементи необхідно обчислювати на льоту, наприклад джерела небезпеки, конфлікти і найближчі загрози. Найпростішою формою штучного інтелекту є система на основі правил, такі зазвичай є фундаментом ШІ. Набір заздалегідь заданих алгоритмів визначає поведінку ігрових об'єктів. З урахуванням різноманітності дій кінцевий результат може бути неявною поведінкової системою, хоча така система насправді зовсім не буде дійсно інтелектуальною. Класичним прикладом гри, де використовується така система, є Pac-Man. Гравця переслідують чотири привиди. Кожен привид діє, підкоряючись простому набору правил. Один привид завжди повертає вліво, інше завжди повертає вправо, третій повертає в довільному напрямку, а четвертий завжди повертає в бік гравця. Якби на екрані привиди з'являлися по одному, їх поведінка була б дуже легко визначити і гравець зміг би без складнощів їх уникати. Але оскільки з'являється відразу група з чотирьох привидів, їх рухи здаються гравцеві складним і скоординованим процесом вистежуванням. Насправді ж тільки останній з чотирьох привидів враховує розташування гравця. З цього прикладу випливає, що правила ШІ не обов'язково повинні бути жорстко заданими. Вони можуть грунтуватися на сприйманні стану ігрового середовища або на параметрах об'єктів. Такі змінні, як рівень агресії, рівень сміливості, дальність огляду та швидкість мислення, дозволяють отримати більш різноманітну поведінка агентів навіть при використанні статичних систем на основі правил. Системи на основі правил є найпростішим варіантом реалізації структури ШІ, так як не застосовують інтелектуальних механізмів і лише створюють видимість «інтелекту». Навість у більш складних ігрових системах в якості основи ШІ, зазвичай, використовуються низки умовних переходів для вибору дії у поточному стані. У тактичних іграх такі правила керують вибором використовуваної тактики, поведінкою окремих NPC-агентів. В стратегічних іграх системи правил використовуються для управління глобальними рішеннями, як послідовність об'єктів для побудови, реакція на конфлікти, і.т.д.

2 ПОНЯТТЯ АДАПТИВНОГО ШТУЧНОГО ІНТЕЛЕКТУ

У попередньому розділі описуються методи проектування систем інтелекту, що вписуються в заздалегідь задані ігрові події. Для багатьох ігор таке рішення цілком можна застосувати, якщо всі спроектовані моделі опрацьовані досить повно і існує чітке розуміння цілей, які переслідують керовані штучним інтелектом об'єкти. Якщо ж в гра потребує значної різноманітності, то у гравця повинен бути сильніший противник, відповідно, ШІ повинен мати здатність пристосовуватися і розвиватися.

Такий ШІ можна використовувати у різноманітних тактичних і стратегічних іграх зі багатопараметричною механікою та великою кількістю можливостей у ігровому процесі. Якщо потрібно зробити гру складною і захоплюючою, такою, щоб гравець не зміг рано чи пізно здогадатися про єдину оптимальної стратегії для перемоги над комп'ютером, ШІ повинен вміти вчитися і пристосовуватися.

Основною функцією, яку повинен виконувати адаптивний ШІ, є корекція ігрового процесу згідно до навичок гравця для досягнення оптимального рівню складності. Складність гри є надзвичайно важливим фактором, що визначає наскільки задовільним виявиться ігровий процес. Як висока, так і низька складність можуть призвести до роздратування гравця і негативного досвіду, при чому «правильний» її рівень є сугубо особистим питанням. Зазвичай з цією метою більшість ігор на початку пропонують вибір рівню складності, але, насправді, такий підхід доволі часто виявляється безплідним[6].

Адаптивні механіки дозволяють зробити ШІ більш схожим на живого супротивника. Такий підхід призводить до зниження вродженої складності, але підвищує набуту. Для різних типів гравців це може виявитися як позитивним, так і негативним результатом, тому перед застосуванням адаптивного ШІ слід проаналізувати тип запланованої гри та цільової аудиторії щоб переконатися у доцільності рішення.

Тим не менш, не дивлячись на переваги застосування адаптивного ШІ, у сучасній ігровій індустрії можна помітити тотальне панування простих ШІ на основі дерев рішень. Така тенденція пов’язана з наступними причинами:

- Звичайний ШІ легше тестувати через його передбачуваність.
- Неочевидність способів демонстрації гравцю природи ШІ з яким він грає, що може призвести до того, що деякі люди не помітять різниці.
- Небезпека «перенавчання», при якому конкретному гравцю може стати набагато легше перемогти адаптивний ШІ на чужому комп'ютері, аніж на своєму.
- Недостача контролю над роботою такого ШІ. Ігрові дизайнери, впроваджуючи різноманітні ігрові сценарії часто хочуть бути впевнені, що в деякій ситуацій ШІ поведе себе певним чином, реалізувати що неможливо в такому випадку, якщо логіка роботи агенту представлена вагами нейронної мережі, або іншою системою, у якій регулятори не піддаються формалізації.

Для великих ігрових компаній будь-які експерименти можуть бути доволі небезпечними, тому що вони вкладають багато ресурсів у розробку своїх продуктів. Тим не менш, корисний ефект, який пропонують адаптивні системи, значно перевищує проблеми, що пов'язані з його реалізацією у грі. Наприклад, проблема тестування значно спрощується якщо комбінувати адаптивні механіки з класичними рішеннями у області ігрових ШІ (що також зменшує вартість розробки). Супроводження набуття досвіду ШІ різноманітними реакціями користувацького інтерфейсу та наративними елементами надасть гравцю уявлення про природу ШІ, з яким він має справу. Якщо здатність системи навчатися природно вписується у ігровий дизайн, гравцю буде цікавіше чекати нових сюрпризів, які можуть підносити адаптивні системи. Небезпека «перенавчання» не є такою страшною, якщо згадати що грати на чужій платформі може бути нецікаво і з других причин: доступ до інших ігрових предметів, інший клас та рівень персонажу, інший прогрес у кампанії і.т.д. Все це є результатами прояву особистості гравця у процесі гри, так само як і результати навчання ШІ. Як матеріал для навчання ШІ може використовувати дані про свої попередні дії та відповідні дії гравця, який був результат так наскільки він відхиляється від бажаного. У статистику гравця може входити час, який він витрачає на рішення певних проблем, улюблені засобі, рівень агресії і.т.д.

Здатність точно передбачати наступний хід супротивника є важливим фактором адаптивної системи. Для виконання певного рішення можна використовувати різні

методи, наприклад розпізнавання закономірностей у минулих діях і рішеннях гравця або ж використання випадкових здогадок. Одним з найпростіших варіантів адаптації є відстеження рішень, що були прийняті раніше, та аналіз міри їх успішності. Система штучного інтелекту має реєструвати певні рішення, що були зроблені гравцем у минулому, зберігати ці дані та оцінювати їх (наприклад, в бойових іграх в якості фактору успішності можна використовувати затрачений на проходження рівню час, втрачене здоров'я чи точність стрільби). Можна збирати додаткові відомості про ситуацію, щоб утворити контекст для рішень, наприклад відносний рівень здоров'я, колишні дії і положення гравця на рівні (тактика гравця може варіюватися в залежності від обставин). Коли найбільш вдалий паттерн поведінки буде знайдено, система підтримуватиметься нього до тих пір, доки він не перестане бути ефективним. Якщо гравець знайде спосіб протидіяти тактиці, система прискорить процес адаптації. Для отримання потрібного результату можна оцінювати історію ігрових подій для визначення рівню успішності колишніх дій і прийняття рішення щодо необхідності зміни тактики.

В стратегічній грі дані про попередні бої гравця можуть допомогти обрати найбільш оптимальну тактику (або створити нову, комбінуючи стандартні пат терни поведінки) для використання неї проти гравця. Наприклад штучний інтелект може повністю на оборону, обрати наступальну тактику, жорстко атакувати всіма силами незважаючи на втрати або ж обрати збалансований підхід. Таке рішення можна ефективно зробити відштовхуючись лише від досвіду декількох попередніх боїв, які дадуть знати про слабі сторони гравця. Так само, у стратегічній грі можна підбирати оптимальний набір різних бойових одиниць в армії проти конкретного гравця. У іграх, де ШІ керує персонажами, що підтримують гравця адаптивний ШІ зможе краще пристосуватися до природного стилю гравця, вивчаючи його дії та визначаючи рівень того, наскільки він потребує допомоги, щоб не втручатися у проходження гри занадто неорганічно, або ж частіше допомагати слабшим гравцям.

3 ВИМОГИ ДО ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ

Виходячи з поставленої мети, розроблювана методологія має дозволяти створювати ігрові ШІ, що можуть пристосовуватися до динамічного середовища, обираючи логічні, з точки зору правил гри, рішення в умовах непередбачених розробником станів системи. Слід зазначити, що, в даному випадку під динамічним середовищем розуміється ігрове середовище, в якому можуть змінюватися правила та параметри протягом циклу розробки. В звичайних умовах це тягне за собою перепрограмування штучного інтелекту у відповідності з оновленням ігрових механік, але адаптивний ШІ має бути здатний адаптуватися до нових умов. Оскільки, на відміну від академічних проектів, в даному випадку система не матиме даних для тренування, мають бути окреслені засоби збору інформації, що буде використано штучним інтелектом для прийняття подальших рішень. Варто зазначити що існує ще одна велика відмінність між штучним інтелектам у відеоіграх та у наукових програмах. Метою наукової програми є певний результат. Якщо розробник ставить за мету створити ідеальну програму для гри в шахи, то час, який програма витрачатиме на хід, буде обмежений лише правилами шахів. Головне – зробити найкращій хід з усіх можливих і для цього розробник буде застосовувати ті ресурси, які порахує необхідними. Комп’ютер для гри в шахи «Deep blue», розроблений IBM та здобувший славу за перемогу над Гаррі Каспаровим, складався з двох стійок з тридцятьма вузлами на кожній. Це буда відносно велика машина, через що навіть виник стереотип, що комп’ютери перемагають людей у іграх лише за допомогою «грубої сили», аналізуючи абсолютно усі можливі варіанти розвитку партії. Потужний суперкомп’ютер Deep Blue в середньому 126 мільйонів позицій у секунду. Максимальна встановлена швидкість під час партій склала 330 мільйонів позицій у секунду.

Коли мова йде про відеоігри, одне з найважливіших питань яке постає – це затрати ресурсів, які застосовує програма. Більшість гравців надзвичайно критично відноситься до вимог гри до комп’ютеру, адже якщо комп’ютер недостатньо потужний, це призведе до низької продуктивності роботи програми, зменшення

кадрової частоти гри, вильотів, перенагрівання відеокарти і, навіть, критичних помилок. Розробники ігор, зазвичай, прикладають усі сили для максимально можливої оптимізації з метою задоволення потреб своєї аудиторії.

Більшість ігор має підтримувати декілька штучних інтелектів одночасно – це можуть бути окремі персонажі, системи, групи ворогів та навіть цілі армії. Слід очікувати низької продуктивності від гри, кожен персонаж якої обробляє 126 мільйонів операцій в секунду для пошуку оптимального варіанту дій, як це робив «Deep Blue». І це не говорячи про те, що більшість сучасних ігор уявляє з себе набагато складніші системи аніж шахи. Гра може моделювати величезні простори с великою кількістю об'єктів та сотнями параметрів для аналізу. Слід врахувати і те, що гра потребує складних обрахувань для відтворення графіки і фізичної поведінки об'єктів і стане зрозуміло чому розробники, як правило, відмовляються від додавання складних систем у структуру гри. Таким чином, для встановленого завдання постає критична умова – переконатися що розроблений ШІ не буде займати занадто багато ресурсів відповідно до рівня складності ігрової системи.

Ще одним важливим критерієм якісного штучного інтелекту є можливість його модифікації з метою реалізації певних ігрових механізмів, які прагне встановити розробник. Використовувані у даній сфері дерева рішень дозволяють точно задати бажану поведінку агентів для різноманітних випадків в той час, коли рішення на основі нейронних мереж, що використовуються у академічній сфері, уявляють з себе «чорний ящик». Розробник даної системи може спостерігати логіку прийняття рішень системою лише як сукупність вагів, що були встановлені у ході тренування, але при цьому не зможе внести зміни до них, які призвели б до довільного бажаного ефекту. В такому випадку єдиним варіантом залишається перетренування мережі за нових правил, що є затратним за часом процесом, який може повторюватися велику кількість разів у процесі розробки. Таким чином, ШІ створюваний на основі розроблюваної методологією повинен бути достатньо гнучким для доповнення будь-якими бажаних розробником паттернів поведінки.

4 ВІДОМІ РІШЕННЯ У ВІДЕОІГРОВІЙ СФЕРІ

Хоча застосування засобів машинного навчання у ігрових системах не є поширеною практикою, можна привести набір прикладів, які в тій чи іній мірі змогли застосувати дану технологію. В даному розділі наведено перелік деяких з даних проектів з метою аналізу рішень, до яких вдались розробники з метою реалізувати адаптивний механізм як ігрову механіку.

4.1 Echo: Ex-Hitman [7]

Echo: Ex-Hitman – це відеогра у жанрі «stealth-action», випущена студією «Ultra Ultra» у 2017-му році. У цій грі гравець має пройти крізь серію локацій, заповнених ворогами, що полюють на нього. Особливість гри є те, що NPC здатні спостерігати за діями гравця і вчитися на їх основі. Протягом гри циклічно змінюються 2 фази: з увімкненим світлом та вимкненим. Будь-які дії гравця протягом освітленої фази можуть бути використані місцевим ШІ для навчання агентів. Після того, як гравець виконає певну кількість дій, пам'ять ШІ буде вважатися заповненою, і гра перейде до темної фази, щоб використати отримані навички у наступній фазі. У темряві ШІ не може спостерігати за гравцем і, відповідно, вчитися у нього. Таке рішення було прийнято, щоб не робити гру занадто стресовою, даючи гравцю проходити частини рівнів не хвилюючись про те, що його дії будуть використані проти нього. Ігрова механіка Echo є чудовим прикладом балансу між дослідженням та експлуатацією середовища, у якому джерелом тренувальних даних є сам гравець. На кожній новій ітерації агенти не лише починають грати все ліпше, створюючи зростаючу криву складності, але й вчяться протидіяти різноманітним стратегіям гравця, попереджаючи знайдення слабкого міста у поведінці агентів, яке гравець зміг би експлуатувати до самого кінця гри, як це можливо зі ШІ на основі дерев рішень. Ще більше уваги заслуговує наявність скритої ігрової механіки, оскільки гравець здатний використати здатність агентів навчатися проти них самих, спеціально виконуючи помилки на різних етапах гри, які ШІ може повторити на наступних ітераціях. Ця можливість

поглиблює ігровий процес, роблячи його менш постійним і створюючи ефективну платформу для появи незаскриптованих ігрових сценаріїв. Розгляд проекту поскладнюється тим фактом, що точно невідомі засоби машинного навчання, які були використані у розробці гри. Для того, щоб NPC могли повторювати прості дії гравця (біг, присідання, стрільба) достатньо застосувати машину станів, яка буде розблоковувати нові вміння агентів у випадку виконання події використання цього вміння гравцем. Оскільки у Echo ворожі агенти здатні знаходити послідовності рішень, що дозволяють протидіяти різноманітним стратегіям гравця, можна припустити що було застосовано елементи машинного навчання на основі винагороди.

4.2 Black & White [8]

Black & White – відеогра у жанрі «симулятор бога», розроблена Lionhead Studios і опублікована Electronic Arts для Microsoft Windows в 2001 році і Feral Interactive в 2002 році для Mac OS. Black & White поєднує в собі елементи симулятору життя і стратегії. Гравець виступає в ролі бога, метою якого є перемога над Немезісом - іншим богом, який хоче захопити світ. Основною темою є концепція добра і зла, при цьому атмосфера впливає на моральний вибір гравця. Головна механіка гри Black & White - це взаємодія між гравцем і істотою, під назвою «аватар», який виконує інструкції гравця. Гравець має можливість навчити свого аватара що і коли їсти, як боротися з противниками, як поводити себе в різноманітних ситуаціях і.т.д. Навчання виконується за допомогою системи на основі навчання з підкріпленням: якщо аватар проявляє небажану поведінку, його можна знеохотити покаранням. Якщо істота виконує дії, які влаштовують гравця, він може заохотити аватара. Система зберігає реакції гравця на різні дії і поступово змінює свою поведінку. Після тривалого тренування істота зможе виконувати комплексні функції та стане повноцінним аватаром. Слід відмітити, що гра демонструє результати процесу навчання не лише діями аватару, але й візуально, змінюючи його зовнішність, таким чином підсилюючи зворотний зв'язок між грою и гравцем, доповнюючи ігровий досвід.

4.3 Forza Motorsport 5[9]

Forza Motorsport 5 – відеогра про автомобільні перегони у жанрі «автосимулятор», що розроблена компанією «Turn 10 Studios» і видана «Microsoft Studios». Гра спрямована на високий рівень деталізації поведінки авто на дорозі та правдоподібну фізику, але є відомою не лише за це. Для реалізації віртуальних супротивників у перегонах, розробники застосувало незвичне для жанру рішення. Зазвичай, автомобільні боти у іграх керуються спеціальними скриптовими алгоритмами, які надають їм більш-менш правдоподібну поведінку: вони здатні дотримуватися дороги, вписуватися у складні повороти і навіть іноді атакувати гравця для отримання переваги у гонці. Тим не менш, все ці можливості є заздалегідь прописаними у коді гри і перевершити власний рівень такі боти не здатні. Forza Motorsport 5, на відміну, застосовує систему «Drivatar» на основі нейромереж для навчання ботів. Працює це наступним чином. Протягом кожної гонки Forza Motorosport 5 записує всю інформацію про те, як ви граєте, на вашу консоль. Система запам’ятовує де ви були, на якій машині їхали, як ви використовували газ та повороти, де були інші машини на трасі та інші деталі. Окрім даних про вашу машину гра також стежить за всією пов'язаною інформацією, встановлюючи співвідношення між вашими діями і діями інших машин на трасі. Це може допомогти відповісти на питання про те, коли і чому гравці роблять певні дії. Наприклад: навіщо та при яких обставинах вони виїжджають за межі траси, у яких ситуаціях гравці вдаються до агресивних стратегій, так як тактика гравця пов’язана з його позицією у перегонах.

Усі ці дані є важливими для навчання системи та надання їй навичок, схожих з людськими. Після кожної гонки консоль гравця завантажує кілька кілобайт даних про змагання на сервери Microsoft, де алгоритми Drivatar невпинно шукають характерні шаблони у масивах даних. Це дозволяє комп’ютеру постійно вивчати нові маневри та поводити себе майже як справжній водій під час гри, демонструючи хитрі прийоми, якім він навчився у живих гравців. Неможливо досягти такого ефекту без

застосування адаптивних систем, хіба що усі маневри заздалегідь будуть внесені у скрипт штучного інтелекту.

Тим не менш, існує ряд проблем з якими зіткнулися розробники «Drivatar» після офіційного виходу гри. Деякі з гравців навмисне демонстрували під час гри погані рішення для того, щоб навчити систему неправильно та познущатися з неї. Для того щоб запобігти негативному ефекту від таких дій, розробники власноруч керують процесом навчання, відсіваючи невдалі тактики, яким навчилася система.

4.4 Left4Dead [10]

Left4Dead – багатокористувацька гра у жанрах «FPS» та «survival horror». Однією з особливостей гри є додатковий ШІ, який тут називається «Режисер» (AI Director). Режисер здатний впливати на ігровий процес, слідкуючи за статистикою успішності гравців та підлаштовуючись під їх рівень.

Режисер слідкує за такими параметрами успішності, як час потрібний команді на проходження одного рівню, середня кількість втрат серед гравців, точність стрільби, середній рівень здоров’я для кожного гравця, кількість ресурсів що застосовуються гравцями. Режисер розміщає комп’ютерних ворогів в залежності від стану і місця розташування гравців, тим самим, намагаючись створити нові враження при кожному проходженні рівня. Візуальні ефекти, музика, погодні умови - все контролюється режисером з метою створення якісного емоційного напруження. Окрім цього, ШІ регулює фази рівню активності гри для запобігання перенапруження гравців, створюючи невеликі перериви між атаками ворогів у випадку якщо гравці демонструють невисокі результати. Схему виконанням режисером функції популяції локацій ворогами наведено на рисунку 4.4.1.

Окрім наративних ефектів, режисер також здійснює контроль над кількістю та різновидами зброї та ресурсів що надаються гравцям, а також може навіть впливати на архітектуру локацій, змінюючи маршрути для проходження. Комплексний підхід до реорганізації структури рівню потребував би багато зайвої роботи художників,

тому система обмежуються просто розставленням перешкод у різних місцях, блокуючи певні альтернативні шляхи.

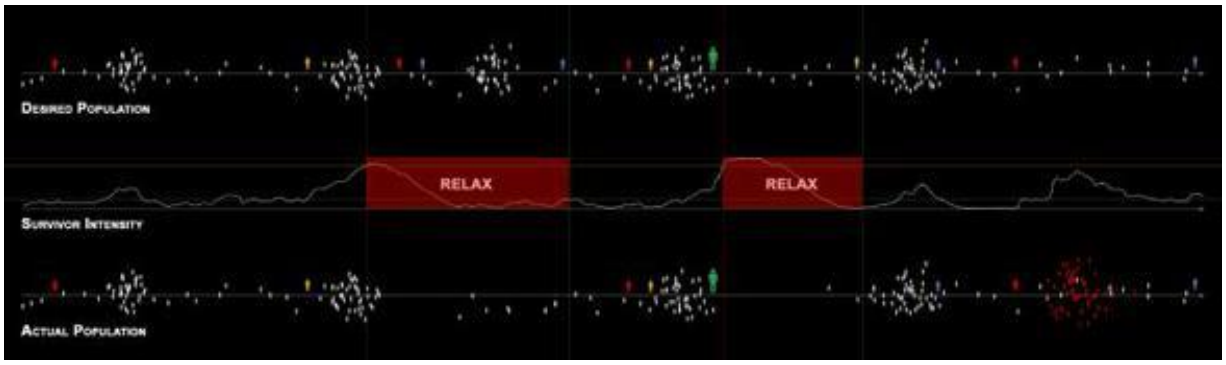


Рисунок 4.4.1 – Графік популяції локації комп’ютерними супротивниками. Червоними прямокутниками відмічені періоди відпочинку, які почалися внаслідок перенапруження гравців

Використання цієї системи робить проходження одних й тих самих рівнів несхожими, що позитивно впливає на реіграбельність. Така риса є особливо цінною для багатокористувацьких ігор, адже на відміну від одиночних, такі ігри розроблюються щоб люди періодично поверталися в них і підтримували задовільний рівень онлайн-активності необхідний для сервісу.

4.5 Створення мап для DOOM з генеративними змагальними мережами [11].

Технології процедурної генерації ігрових рівнів існують вже кілька десятиліть, зазвичай їх реалізують випадковим чином об’єднуючи створені вручну частини локацій у одну цілу [12]. Дослідники із Politecnico di Milano застосували генеративні змагальні мережі для генерації локацій для шутеру від першого обличчя «Doom», що було випущено у 1993 році. Гра має вбудований редактор локацій, який було використано у зв’язці з розроблюваною системою. Генеративні змагальні мережі - це клас алгоритмів машинного навчання, що використовуються в навчанні без учителя, реалізовані на основі системи з двома нейронними мережами, які змагаються одна з одною в рамках гри з нульовою сумою [13]. Дослідники використовували нейронні

мережі для аналізу існуючих карт DOOM, а потім генерували нові карти, подібні до оригіналів Вони розглядали два типи GAN при створенні нових рівнів: варіант, що тільки використовував зовнішній вигляд навчальних карт, і інший, який використовував як зовнішній вигляд, так і метрики, такі як кількість кімнат, довжина периметра, тощо. Обидві мережі показали спроможність створювати якісні локації. Розробники відмітили, що хоча система і здатна створювати локації подібні до тих, що створюються людьми, багато деталей і нюансів, що використовуються дизайнерами було упущено нейронними мережами через відсутність достатньої кількості якісних метрик. Приклади рівнів, що були створені нейронною мережею наведено на рисунку 4.5.1. Система дозволяє не лише отримати багатоповерхову архітектуру локації, але й заповнити її інтерактивними об’єктами, ресурсами та ворогами.

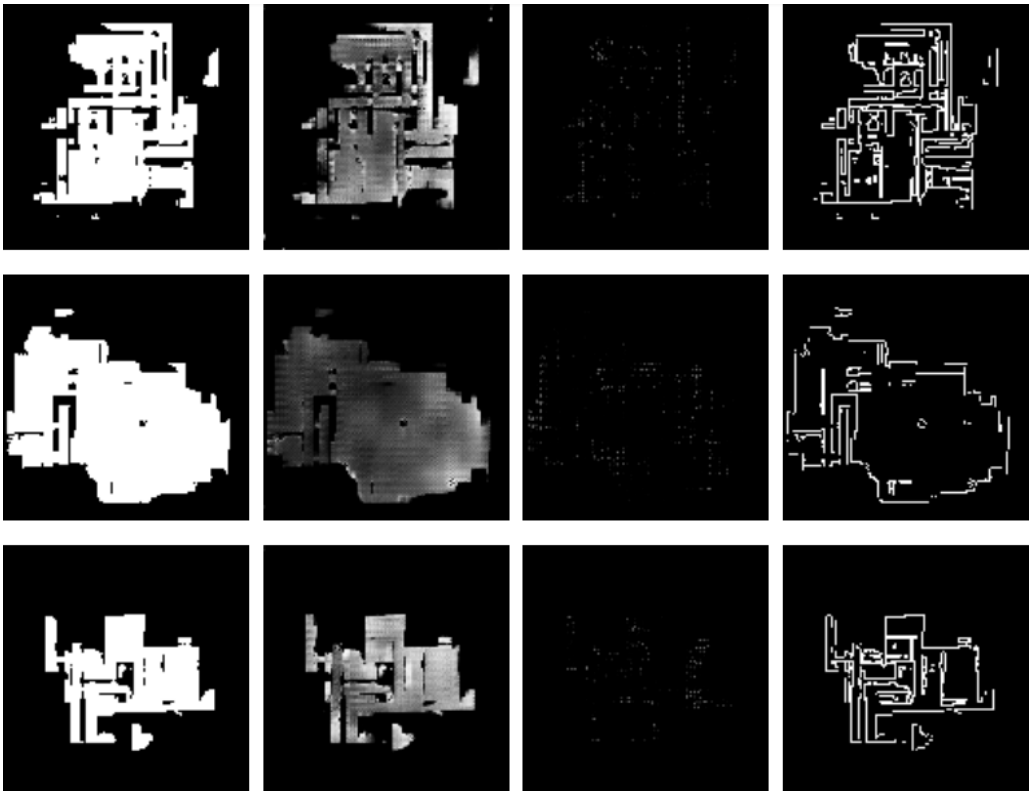


Рисунок 4.5.1 - Приклад карт DOOM, створених системою. Кожен рядок є однією картою, а кожне зображення - один з аспектів карти (поверхня, висота, об’єкти та стіни)

4.6 Динамічне балансування складності[14].

Метод динамічного балансування складності – це процес автоматичної зміни параметрів, сценаріїв та поведінки в відеогрі в режимі реального часу, заснований на навичках гравця, щоб уникнути відчуття нудьги у гравця (якщо гра дуже легка) або розчарування (якщо вона занадто складна). Різні люди ставлять перед собою різні цілі, граючи в ігри. Деякі гравці бажають таким чином просто відпочити, інші – хочуть зіткнутися зі змаганням. Застосування деяких ігрових механік може дозволити краще пристосуватися до особистих інтересів гравця. При цьому, ця адаптація не обов'язково може стосуватися штучного інтелекту, з яким взаємодіє гравець, а й усього ігрового середовища і навіть правил гри.

Знімаючи деякі обмеження з комп'ютерних противників, гра може дозволити їм шахраювати, наприклад, ШІ-супротивникам у перегонах може бути надана необмежена швидкість, щоб постійно залишатися поруч з гравцем, підтримуючи так рівень напруги. Мета динамічного балансування складності - утримання користувача зацікавленим від початку до кінця, забезпечуючи хороший рівень виклику.

Традиційно, складність гри неухильно зростає по ходу гри (або в гладкому лінійному способі, або за допомогою кроків, представлених рівнями складності). Часто ігри у самому початку пропонують гравцям обрати рівень складності. Проте, це може виявитися гнітючим як для досвідчених, так і недосвідчених гравців, так як, обираючи певний рівень складності, вони можуть пізніше усвідомити що їх вибір не задовольнив їх очікування. Динамічне балансування складності усуває цю проблему, створюючи індивідуальний досвід для кожного гравця. Як користувач поліпшує свої навички з плином часу, так і рівень проблем постійно зростає. Однак реалізація таких елементів створює багато труднощів для розробників ігор. В результаті, цей метод не отримав широкого розповсюдження.

Прикладом такого різновиду адаптивності є гра Resident Evil 4, яка застосовувала систему «шкали складності», про існування якої гравці не повідомлялись. Ця система слідкувала за успішністю гравців, оцінювала її по шкалі від 1 до 10 та змінювала поведінку супротивників відповідно до значення

5 ВІДОМІ РІШЕННЯ У АКАДЕМІЧНІЙ СФЕРІ

5.1 Застосування глибокого навчання з підкріпленням для гри на Atari

Наприкінці 2013 року тоді компанія DeepMind досягла прориву у машинному навчанні: за допомогою глибокого навчання з підкріпленням вони розробили систему, яка могла навчилася грати у більшість класичних ігор Atari на людському рівні та вище[15]. Вони розробили нейронну мережу, що виступала у ролі функції якості Q-навчання, входом якої були пікселі з монітору, а виходом – один з сигналів управління джойстиком від Atari. Зазвичай при Q-навчанні застосовується таблиця, яка характеризує функцію $Q(s, a)$ якості певної дії у певному стані. Оскільки система вчилася грати без можливості доступу до ігрових змінних, піксели були єдиним джерелом даних для тренування, але з цієї причини кількість можливих станів виявилася завеликою для формування таблиці, тому для апроксимації значень пар стану-дії було використано нейронну мережу, структура якої зображена на рисунку 5.1.1

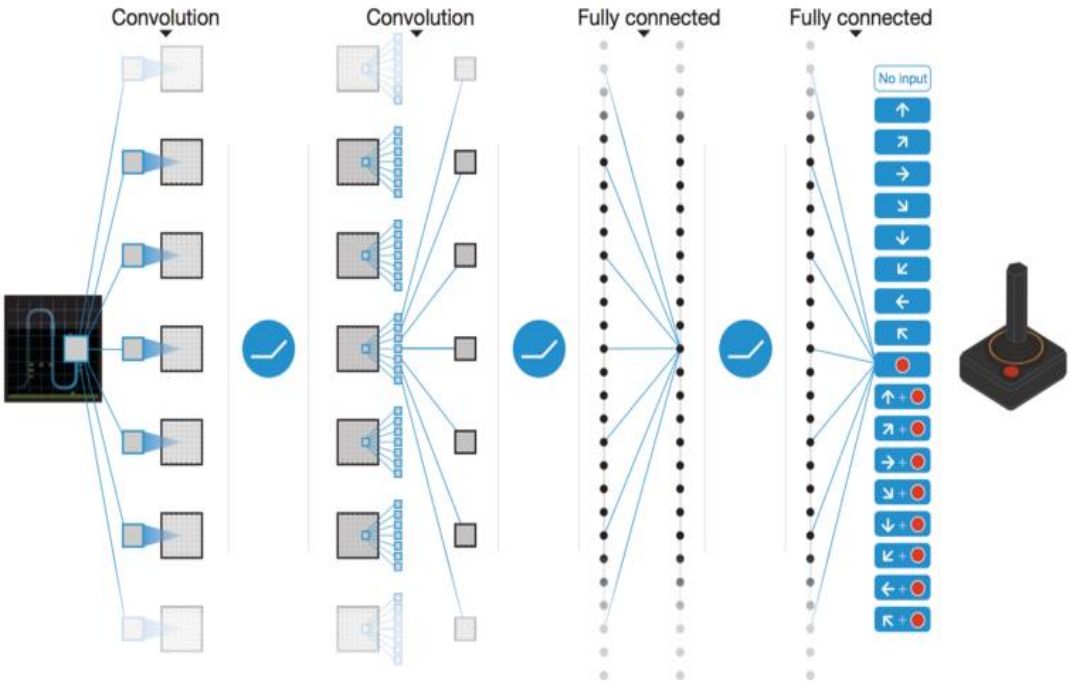


Рисунок 5.1.1

Таким чином, агент на основі навчання з підсиленням має доступ тільки до тієї інформації, яка надається і реальним гравцям. З одного зображення агент RL може отримати дані про поточні позиції ігрових об'єктів, але, комбінуючи поточне зображення з кількома попередніми, нейронна мережа здатна визначити не тільки позиції, але і фізичні характеристики ігрових об’єктів, такі як швидкість та прискорення завдяки різниці між послідовними кадрами гри (Рисунок 5.1.2)

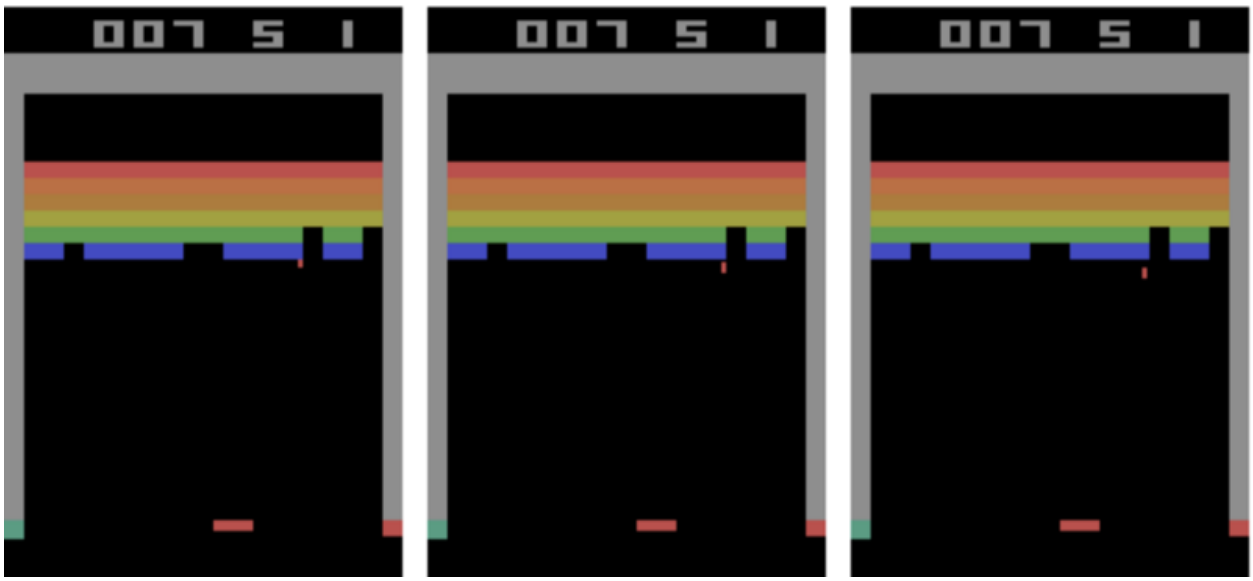


Рисунок 5.1.2 – Кадри гри Breakout. Керуючись одним зображеннями, можна визначити позицію м’ячика. Двох достатньо для знаходження напряму та швидкості його руху. З урахуванням трьох кадрів можна визначити прискорення.

Принцип навчання з підкріпленням полягає у виборі однієї з можливих дій виходячи з поточного стану на основі максимізації очікуваної винагороди. Таким чином, система не потребує дані для тренування, оскільки тренується у процесі взаємодії зі своїм середовищем шляхом спроб і помилок. Отримуючи позитивну винагороду за певну дію, система підвищує ймовірність повторювання даної дії в наступний раз коли опиниться у тому самому стані, в той час як негативна винагорода зменшить цю ймовірність. У випадку експерименту з Atari тренувальний процес починається з того, що агент випадково вибирає одну з можливих дій, після чого

отримує винагороду та наступний станом. Дані цього переходу потім збираються в кортежі, як скаладаютья з полів: стан, дія, винагорода, наступний стан. Кортеж зберігається в пам'яті, яка зберігає лише певну кількість останніх переходів. Після того, як агент зібрав достатній досвід, модель поведінки закріплюється. Під час тренування, кожну ітерації, агент приймає випадкове рішення з деякою ймовірністю ε. В іншому випадку стан передається в нейронну мережу, і агент обирає дію, за яку він очікує отримати найвищу кумулятивну винагороду. Коефіцієнь ε змінюється лінійно від 1,0 до 0,1 протягом мільйона ітерацій, після чого залишається рівним 0,1, як показано на рисунку 5.1.3

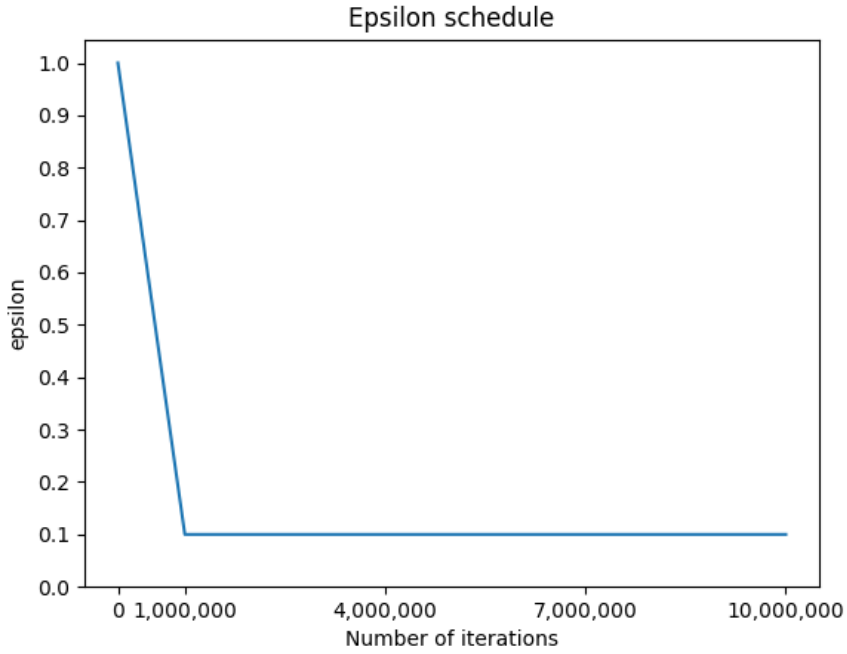


Рисунок 5.1.3 – Зміна ε протягом процесу тренування

Це означає, що на початку навчального процесу агент переважно досліджує середовище, але після накопичення досвіду починає експлуатувати його. У [15] Google Deep Mind наводить статистику результатів тренування на різних іграх, що приведено на рисунку 5.1.4

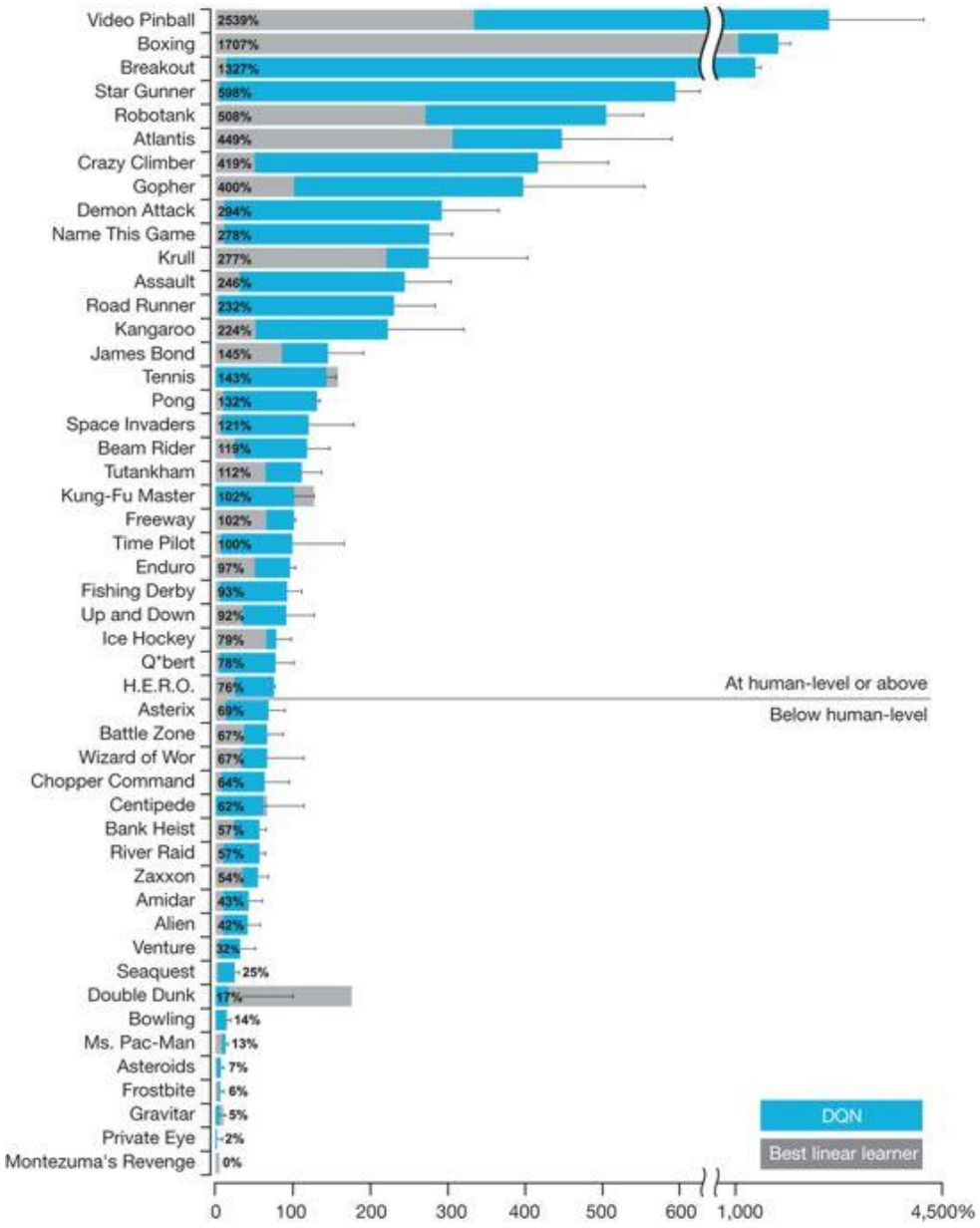


Рисунок 5.1.4 – Результати тренування із підкріпленням у різних іграх Atari

Згідно до [15] система потребує, в середньому, близько 2 годин для того щоб натренуватися грати у одну з ігор на людському рівні. Це високий результат, особливо, враховуючи що у сфері розробки ігор надзвичайно рідко постає задача створення ШІ, який перевершує здатності людини, що відмітає необхідність подальшого тренування. При цьому, слід відмітити, що агенти тренуються в умовах незмінного середовища, де не існує можливості виникнення трансформацій, що можуть призвести до нівелювання накопиченого досвіду. Також, важливо врахувати що алгоритм тренується на старих іграх, які мають значно меншу кількість ігрових механік, умовностей керування та стратегічного потенціалу аніж сучасні.

5.2 AlphaGo [16].

AlphaGo – це комп'ютерна програма для гри в ґо, що розроблена компанією Google DeepMind. У жовтні 2015 року вона стала першою комп'ютерною програмою, що в грі на рівних на класичній дошці 19x19 завдала поразки людині – професійному гравцю в ґо. Гра Го завжди була міцним горішком для ІІ. Скептики стверджували, що програми в го або ніколи не переможуть, або отримають перемогу дуже нескоро. Го складніше шахів в 10^{100} разів - саме в стільки разів більше можливих позицій каменів на стандартній дошці 19 × 19, ніж в шахах. Це ускладнює використання традиційних комп'ютерних методів, наприклад методу повного перебору. Після перемоги комп'ютера Deep Blue над чемпіоном світу з шахів Гаррі Каспаровим 1997 року, штучному інтелекту знадобилось майже 20 років щоб зрівнятися з гравцями в ґо.

AlphaGo досконаліша за інші програми для гри в ґо. У 500 Іграх проти других програм, враховуючи Crazy Stone та Zen, AlphaGo перемогла 499 разів . У жовтні 2015 року, AlphaGo перемогла трьохразового чемпіона Європи Фаня Хуея з рахунками 5:0. Це перший випадок, коли комп'ютерна програма перемогла професійного гравця в ґо на класичній дошці в грі на рівних

У AlphaGo одна з нейронних мереж (value network) використовується для оцінки позицій в термінальних вузлах дерева пошуку, інша (policy network) призначена для розумного відбору ходів-кандидатів. Також використовується невелике дерево перебору Монте-Карло. У кожному вузлі розглядається 1-2 ходи, середній коефіцієнт розгалуження - менше двійки. Завдяки цьому стало можливим порахувати осмислене дерево з дуже обмеженим числом варіантів, і за рахунок цього знайти дійсно найсильніший хід в позиції.

Спочатку AlphaGo тренувалася на партіях найсильніших гравців в ґо. В якості вхідних даних для навчання використовувалися 160 тис. Ігор з 29,4 млн позицій. З моменту, коли вона стала грати не гірше звичайних людей, корпус для тренування нейронної мережі або для побудови параметрів оціночної функції став генеруватися автоматично. Програма почала грати сама з собою, додаючи нові партії в навчальну вибірку.

6 АНАЛІЗ НАВЕДЕНИХ ВІДОМИХ РІШЕНЬ

На основі розглянутих проектів можна побачити значну варіацію у реалізації механізмів адаптивності в залежності від жанру гри. У Echo здатність агентів вчитися є головною особливістю гри, центральною ігровою механікою, яка надає проекту унікальності у порівнянні з іграми аналогічного жанру на ринку. Покупаючи таку гру, гравець в першу чергу очікуватиме побачити ефективній ШІ, з яким буде цікаво змагатися. Це означає що ця гра має відкритий адаптивний механізм, який не є секретом для гравця і експлуатується як частина ігрового процесу. Тому перед розробником постає задача переконатися що незалежно від ігрового стилю, кожен гравець зможе оцінити у повній мірі місцевий ШІ отримати унікальний досвід взаємодії з розробленими адаптивними агентами. Це є неочевидною задачею, яка потребує не лише ретельного тестування, але й візуального підкріплення, яке дозволить гравцю наявно побачити і зрозуміти концепцію «самонавчального» ШІ. Таким чином, реалізація аналогічного механізму є задачею, що надзвичайно залежить від особливостей розроблюваної гри таких як: жанр, геймплей, візуальне оформлення, тому слід очікувати що обрані розробником інструменти реалізації значно варіюватимуться в залежності від проекту. Для виокремлення загальних нюансів розглянемо інші проекти.

У грі Left4Dead немає необхідності робити «розумних» ворожих агентів, які були змогли б вчитися на основі змагання з гравцями з декількох причин. По-перше, на відміну від Echo, ця гра розрахована на декількох гравців, що грають короткими сесіями. Це означає що агенти мали б збирати інформацію про кожного окремого гравця, і зберігати між іграми для того, щоб скомбінувати дані навчання з даними інших гравців. У ході цієї операції унікальні для кожного гравця вироблені властивості ШІ стали б непомітними, що позбавить сенсу застосування даної технології. Другою, більш значною причиною, є жанрові і сюжетні особливості гри, у межах яких здатність агентів вчитися не має сенсу. Натомість, ця гра реалізує більш практичну ігрову механіку, що дозволяє змінювати елементи гри згідно до успіхів та

стилю гравців, що не лише дозволяє покращити ігровий досвід згладивши криву напруження, а й робить перепроходження кожного рівню несхожим на попереднє, що підвищує реіграбельність – важливу характеристику для кооперативної онлайн-гри. Таким чином, «режисер» дозволяє забезпечити дві основні функції: балансування складності і процедурна генерація рівню. Ці адаптивні механізми працюють на основі збору даних про гравців протягом ігрової сесії і, з технічною точки зору, не потребують алгоритмів машинного навчання для реалізації, хоча вони й могли бути використані. Проводячи паралель з Echo, можна відмітити що робота адаптивного механізму, не дивлячись на велику різницю у функціях ШІ даних ігор, базується на наборі аналогічних принципів: сбір даних про гравця у процесі гри за обраними розробником критеріями та застосування отриманих даних у деякій таргетній функції, що присвоює значення параметрам ігрового середовища. Слід очікувати, що даний механізм широко використовується у іграх на рівні експертної системи, коли розробник безпосередньо вирішує як ти чи інші дії гравця впливатимуть на систему, що робить даний підхід аналогічним до застосування статичних алгоритмів, як дерева рішень. ШІ, при застосуванні даного підходу, нескладно тестувати і налаштовувати згідно до побажань геймдизайнера, але, в свою чергу, це може призвести до передбачуваності ігрового процесу і репетативних сценаріїв.

Forza Motorsport 5 є успішним прикладом впровадження нейронної мережі у ігровий процес. Реалізації ШІ шляхом сбору даних з гравців після кожної ігрової сесії та їх застосування для тренування мережі на серверах розробників вирішує набір питань імплементації нейромереж у іграх: мінімізація навантаження на пристрій клієнта, необхідність попереднього тренування. Окрім цього, це рішення дозволяє впровадити механізм, що поглиблює ігровий процес, створюючи нетипову задачу для даного жанру: тренування гонщика-аватара, що допомагає довше підтримувати інтерес гравців. Застосування виділених серверів не завжди можливо у іграх, особливо це має мало сенсу у випадку розробки одиночної гри, що обмежує можливий простір застосування даного прийому. Слід, також зазначити, що в даному випадку тренування нейромережі має сенс, оскільки мета ігрової механіки – створити

ШІ, що керує автомобілем подібно до того, як це робить гравець, але це, в свою чергу, робить складнішим контролювання процесу навчання системи, оскільки такий ШІ є «чорним ящиком» і неможливо змінити ваги нейромережі таким чином, щоб виключити певні шаблони поведінки. Наприклад, деякі гравці навмисне тренували своїх аватарів грати погано, допускаючи помилки і створюючи аварії, що могло негативно вплинути на ігровий процес інших гравців. Цю проблему не було вирішено.

Генерація локацій широко використовується у відеоігрових проектах з метою підвищення реіграбельності, зазвичай представляючи з себе комбінування заготовлених частин локацій випадковим чином за певним набором правил. Користуючись класичними алгоритмами, можливо створювати величезні ігрові світі, які, при цьому, будуть мати реалістичні географічні властивості, як на рисунку 6.1

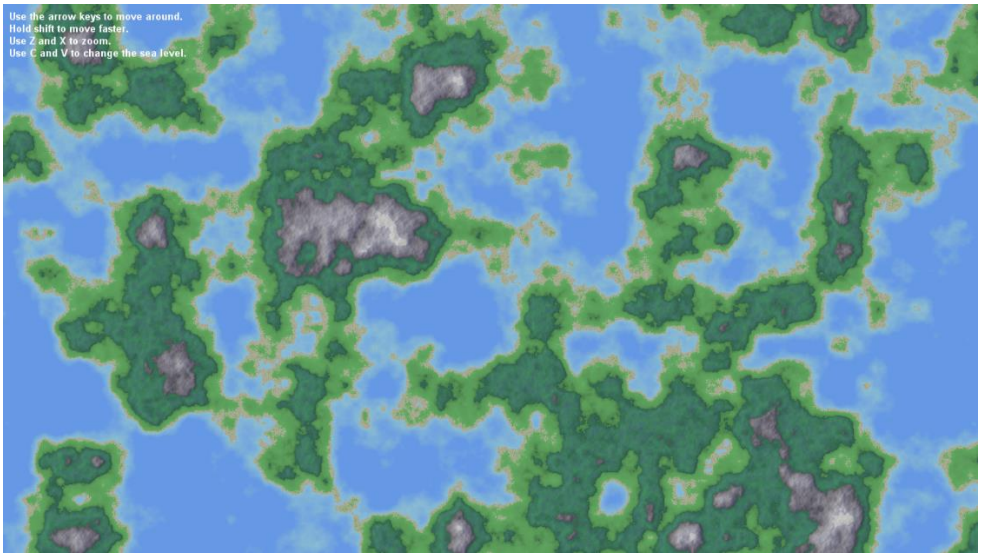


Рисунок 6.1 – Процедурно згенерована локація

Застосування нейромереж дозволяє створювати унікальні локації без використання заготовлених частин, але потребує бази даних для тренування і обмежує контрольованість параметрів результату, що важко піддаються формалізації. Такий підхід можливо використовувати для генерації терейнів для створення унікальних ігрових світів (як для попереднього приготування мапи, так і для процедурного створення під час гри) або окремих ігрових рівнів (ліси, печери, ітд –

локації у яких будуть повторюватися об’єкт і цілі, але які будуть відрізнятися на кожному рівні). Хоча такий підхід і ефективний, але він потребує певної жанрової належності гри, так як у іграх типу «Rogue» гравці очікують побачити випадково створені світи і не будуть звертати увагу на неточності у дизайні рівнів, знаючи що вони створення автоматично, в той час як в одиночних пригодницьких іграх гравець може очікувати побачити ігровий світ, що був створений попередньо. Навіть якщо для цього варіанту було застосовано нейромережу, розробникам все одно доведеться перевіряти результат і, ймовірно, змінювати ті частини, що їх не влаштовують, що показує, що в даному випадку більш логічним рішенням є розроблення бажаного ігрового світу цілком вручну, що знову демонструє перевагу у застосуванні нейромереж для ігор з процедурною генерацією світу, але не для ігор зі статичними локаціями. Це обмежує поле застосування даного методу разом з необхідністю створювати тренувальні дані, з чого можна зробити висновок що слід очікувати надання переваги розробниками класичним методам у даному полі у найближчий час. Результати досліджень Politecnico di Milano мають більше академічне значення аніж прикладну цінність у розробці відеоігор. Цей самий тезис можна віднести до більшості досліджень на тему застосування алгоритмів машинного навчання у ігрових системах, в тому числі і до AlphaZero, оскільки головною метою подібних проектів є дослідження властивостей нейронних мереж для створення ефективних засобів вирішення проблем, що важко піддаються формалізації для подальшого застосування їх у інших сферах. Для того щоб подібний експеримент був об’єктивним, система має бути цілком самонавченою та отримати якнайвищий результат (з точки зору правил гри), щоб можна було говорити про те, що дану технологіях можливо застосувати у інших сферах без експертної підтримки. У відеоігрових проектах ці особливості, зазвичай, будуть недоліками через неможливість контролювати поведінку агентів геймдизайнером та непереможність ШІ. Можливо припустити, що в окремих випадках створити гнучкий ШІ за допомогою нейромереж можливо за допомогою навчання декількох різних мереж з різними налаштуваннями для різних рівней складностей та ігрових режимів, але такий підхід призведе до необхідності перенавчання усіх агентів при кожній зміні

ключових параметрів ігрового середовища, як, наприклад, правил гри. Також слід відмітити, що в той час як більшість академічних проектів використовують у якості ігрового середовища настільні ігри, або прості відеоігри для обмеження середньої кількості можливих дій. У сучасних іграх нерідко агенти мають набагато більшу свободу, так як вони мають орієнтуватися та пересуватися у фізичному трьохвимірному просторі, взаємодіяти з об'єктами що знаходяться на різних позиціях, реагувати на різноманітні подразники у середовищі. Навчати агента взаємодіяти з подібним середовищем лише за допомогою тренування нейромережі є громіздкою та невиправдано складною і довготривалою задачею у порівнянні зі стандартними методами. Наприклад, застосувати відомий алгоритм пошуку шляху (pathfinder) [17] швидше та простіше, аніж тренувати нейромережу доки вона не знайде взаємозвязок між контролем агенту, його позицією у просторі та умовностями руху по ігровому середовищу. Таким чином, для досягнення мети даної дисертації подібний більш гнучкий підход, що не представлятиме (або представлятиме лише частково) результуючий адаптивний ШІ у вигляді «чорного ящику».

Black & White ефективно застосовує навчання з підкріпленням для створення унікального ігрового досвіду. Агенти на основі даного типу машинного навчання виконують випадкові дії і очікують реакції від середовища у вигляді винагороди, яка призводить до зменшення або збільшення ймовірності повторення цієї дії у майбутньому. В даному випадку цим середовищем виступає сам гравець, оцінюючи дії агенту, але с тим самим результатом можливо впровадити систему яка оцінюватиме дії агенту автоматично за деяким встановленим розробником критерієм. В такому випадку розробник може легко налаштовувати бажані параметри поведінки системи і підхід залишиться валідним навіть після введення нових довільних ігрових механізмів у ході розробки. Даний тип навчання відноситься до машинного навчання без вчителя, відповідно він не потребує попередніх даних для тренування, що прискорює процес розробки. Існує два типи задач машинного навчання з підкріпленням:

- Прогнозування – коли система намагається визначити нагороду, яку отримає агент, взаємодіючи зі середовищем
- Управління – коли система приймає рішення, намагаючись максимізувати фінальну винагороду

В даному випадку постає задачі управління, у якій ШІ має приймати рішення щоб досягти максимуму критерія, встановленого розробником ігрового середовища.

За природою задачі машинного навчання з підкріпленням поділяються на епізодичні (Episodic) та неперервні (Continuous). Вирішувана проблема відноситься до класу неперервних завдань, оскільки не наявний кінцевий стан, за якого система зупинить навчання, як у випадку з епізодичними. Прийняття рішень у навчанні з підкріпленням, засновано на принципі максимізації кумулятивної винагороди і, зазвичай, виконується за допомогою деякої функції, що називається політикою та обирається розробником. Ця функція застосовується для вибору бажаної дії a в умовах стану s, виходячи з кумулятивної винагороди R, яку було отримано за застосування даної дії у складі набору рішень раніше. Згідно до поставленої задачі, алгоритм агентів, що використовують подібний підхід, не матиме тренувальних даних і зобов’язаний приймати рішення шляхом спроб і помилок, експлуатуючи зворотній зв'язок із своїм середовищем, таким чином методика навчання з підкріпленням відповідає бажаному підходу. Також, враховуючи наявність явної функції оцінки рішення, можна зробити припущення що дана система може бути налаштована згідно до побажань розробника.

Для використання навчання з підкріпленням необхідно вирішити задачу Марковського процесу прийняття рішень [18]. Марковський процес прийняття рішень використовується для моделювання прийняття рішень в ситуаціях, коли результати є частково випадковими і частково підконтрольні агенту, який приймає рішення. Марковський процес прийняття рішень складається з наступних елементів:

- Множина станів S – це сукупність усіх можливих станів середовища, у яких може перебувати агент.

- Множина дій A – це множина усіх доступних рішень, які може приймати агент протягом свого існування.
- Ймовірність переходу $P\ (s'|\ s, a)$ – це ймовірність переходу до стану s' в момент часу $t\ +\ 1$, якщо було виконано дію a у стані s в момент часу t.
- Функція винагороди $R\ (s'|\ s, a)$ – це функція яка визначає винагороду, яку отримує агент при переході зі стану s до стану s', шляхом виконання дії a.
- Коефіцієнт дисконтування γ – це коефіцієнт у інтервалі від 0 до 1 включно, який визначає до якого типу винагород схиляється агент: до короткотривалих або довготривалих відповідно Винагорода множиться на коефіцієнт дисконтування для попередження зациклення агента, інакше він може завжди обирати винагороду у короткотривалій перспективі і не стане досліджувати середовище для пошуку альтернативних рішень, які можуть принести більшу кумулятивну винагороду.

Марковський процес можна розуміти як сукупність станів S з набором діяй A, можливими у будь-якому стані з певною ймовірністю P. Кожна така дія призводить до деякої винагороди R. Якщо ймовірність і винагорода у такій системі невідомі, то дана проблема належить до проблеми навчання з підкріпленням. Таким чином, задачею машинного навчання з підкріпленням є максимізація винагороди для задачі, описаної через Марковський процес прийняття рішень, де результуюча винагорода без урахування дисконтування визначається як:

$$R_t = \sum_{k=0}^{T} R_{t+k+1}$$

З урахуванням коефіцієнту дисконтування значення результуючої винагороди дорівнює:

$$R_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

Можливими підходами у навчанні з підкріплення є:

- Підхід на основі політики, що застосовує функцію політики a= π(s), яка оптимізується у ході навчання та використовується для вибору дії a у стані s детермінованим або стохастичним шляхом.
- Підхід на основі оцінки, що ставить перед агентом мету оптимізації функції оцінки V(s) яка повертає очікувану винагороду в залежності від досягненого стану.

Політика (π) – функція поведінки агента, яка відповідає за вибір дії відповідно до поточного стану. Політики бувають двох видів:

- Стохастичні: $\pi(a|s) = P(At = a, St = s)$, коли дія обирається випадковим чином с певною ймовірністю $\pi(a|s)$
- Детерміновані: $a = \pi(s)$, коли для кожного стану є лише одна можлива дія

Таким чином, політика застосовується для вибору певної дії із множини можливих дій в залежності від стану. Слід відмітити, що використовується два підходи у імплементації навчання на основі політики:

- Навчання з активною політикою (On policy learning), за якого переоцінка та оптимізація виконується для тієї самої політики, яка використовується у процесі навчання
- Навчання з пасивною політикою (Off policy learning), за якого, у ході навчання, переоцінюється одна політика (таргетна), а для прийняття рішень використовується інша (біхевіоральна).

Функція оцінювання визначає очікувану майбутню винагороду, дотримуючись політики π для кожного стану та визначається як:

$$v(s) = E\,[G_t\,|S_t = s],$$

де G_t – кумулятивна винагорода $G_t = R_{t+1} + R_{t+2} + \cdots + R_{t+n}$

Існує два типи функції оцінювання:

- Функція оцінки за станом $v_\pi(s)$ визначає наскільки бажаним є поточний стан, у якому знаходиться агент і визначається як:

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

- Функція оцінки за дією $q_\pi(s,a)$ визначає наскільки бажаною є певна дія у даному стані і визначається як:

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$$

Для отримання кращого результату, політика має завжди приймати кращі рішення, виходячи з ідеї максимізації винагороди агента. Політика, що дозволяє отримати максимальну можливу винагороду називається оптимальною [19]. Слід зазначити, що таких політик може бути декілька у випадку, якщо вони дають однаковий результат. Оптимальні політики за станом та дією визначаються, відповідно, як:

$$\pi_o(a|s) = \arg\max_\pi V_\pi(s)$$
$$\pi_o(a|s) = \arg\max_\pi Q_\pi(s,a)$$

При рішенні задачі імплементації машинного навчання з підкріпленням постає питання вибору політики, так як вона залежить від особливостей середовища. Оскільки політика обирає оптимальне рішення згідно до значення функції оцінки, значення функції має залежати від кумулятивної винагороди, яка є сумою усіх винагород за ігрову сесію. Враховуюче це, запишемо функцію оцінки як:

$$v(s) = E\ [G_t\ |S_t = s] = E[R_{t+1} + R_{t+2} + R_{t+3} + \cdots |S_t = s]$$

З врахуванням дисконтування, рівняння прийме вигляд:

$$v(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s]$$

Винесемо коефіцієнт дисконтування за дужки і запишемо суму майбутніх винагород як кумулятивну винагороду

$$v(s) = E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)|S_t = s] = E[R_t + \gamma G_{t+1}|S_t = s]$$

Таким чином, значення функції оцінки дорівнює сумі винагороді за виконану дію і функції оцінки наступного стану. Це рівняння називається рівнянням Беллмана [20]. Це рівняння використовується у одному з різновидів машинного навчання з підкріпленням, яке належить до навчання з підходом на основі оцінки і називається Q-навчання (Q-learning). Застосування даного підвиду навчання з підкріпленням передбачає формування деякої функції корисності $Q(s, a)$, якою, зазвичай, виступає так звана «Q-таблиця», що представляє собою матрицю розміром $s * a$. Ця таблиця містить коефіцієнт для кожної пари стан-дія і ініціалізується як нульова матриця, таким чином на початку роботи алгоритму усі можливі рішення мають однакові шанси бути прийнятими. У ході навчання алгоритм ітеративно обирає рішення з найбільшим значенням $Q(s, a)$ та застосовує рівняння Беллмана для оновлення значень Q-таблиці після отримання винагороди R за обране рішення, яке визначається як:

$$Q(s, a) = Q(s, a) + \alpha[R + \gamma * maxQ(s', a') - Q(s, a)],$$

де: α – фактор навчання [0,1], який характеризує пріоритет нової інформації над старою. Оскільки в даному експерименті розглядається динамічне середовище, де винагороди за одні й ті самі рішення можуть змінюватися з ходом часу, α має бути

достатньо високим, щоб не акцентувати увагу на минулому досвіді, який має шанс стати невалідним.

γ – фактор дисконтування [0,1], який характеризує пріоритет майбутніх винагород над короткочасними.

Як видно з формули, рівняння Беллмана було модифіковано відніманням поточного значення Q-функції від максимальної очікуваної винагороди. Це називається «тимчасова різниця», ця операція виконується для того, щоб значення Q-функції демонструвало відносну ефективність рішення, а не абсолютну вираховувану.

Коли можливих станів системи забагато для зберігання таблиці у пам'яті, можна об'єднати Q-навчання з нейронною мережею, яка візьме на себе функцію класифікатору станів, обмежуючі розмір Q-таблиці. Такий підхід називається Deep Q-learning. Подібний підхід було застосовано DeepMind для тренування системи для гри на консолі Atari. Працює це наступним чином: по-перше, визначається політика у вигляді нейронної мережі, яка реалізує механізм прийняття рішень агента. Мережа повинна приймати стан s в якості вхідних даних і виробляти значення Q для кожної дії в множині можливих дій. У випадку експерименту DeepMind вхідним станом були масиви пікселей n останніх кадрів гри. Під час прогнозування натренована мережа використовується, щоб передбачити наступну найкращу дію, для виконання в середовищі, таким чином, агент отримує певний вхідний стан і мережа повертає значення Q для всіх можливих дій, після чого агент приймає рішення, якому відповідає максимальне значення Q, або приймає його с певною ймовірністю, що відповідає співвідношенню значень Q які повернула нейронна мережа. Структура даного механізму прийняття рішень вказана на рисунку 6.2. Слід відмітити, що метою DeepMind було наближення умов тренування алгоритму до процесу тренування реального гравця, тому системі надавалися масиви пікселей у якості станів. У випадку, який розглядає дана дисертація, агент є частиною ігрової системи, він може отримати програмний доступ до необхідних змінних, тому задача розпізнавання не

стоїть, до того ж кількість можливих станів у обраній грі є обмеженою, таким чином виключаючи необхідність імплементації функції розпізнавання в даному випадку.

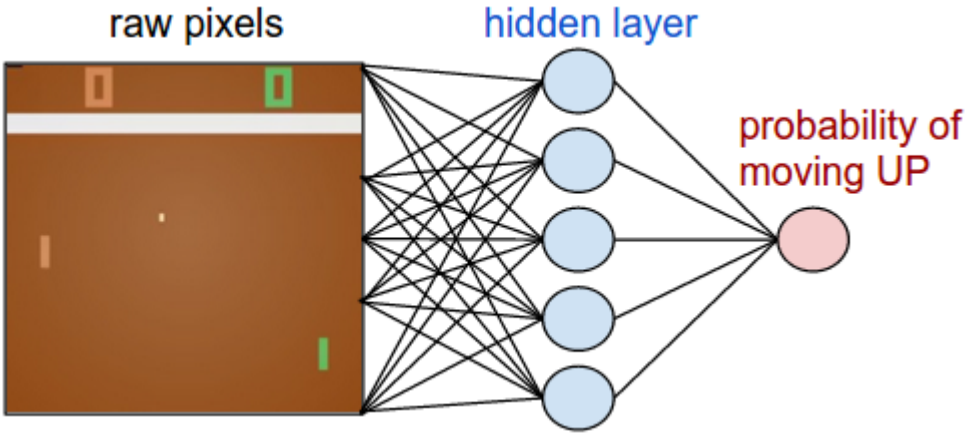


Рисунок 6.2 – Політика прийняття рішень у вигляді двохшарової нейронної мережі

Дослідження показали, що в умовах незмінних правил гри агент ефективно знаходить оптимальну стратегію гри, досягаючи надлюдського рівню[21]. Ці результати несуть важливе значення для академічних досліджень, але незастосовні у сфері відеоігор, оскільки метою розробника є створення штучного інтелекту, достатньо ефективного щоб задовільнити потреби споживачів, але не настільки, щоб він став непереможним, що є ще однією причиною, чому алгоритми машинного навчання не використовуються широко у даній сфері. З технічної точки цю проблему можливо вирішити за рахунок введення деякої ймовірності помилки при виборі рішення, але такий підхід призведе не до зменшення складності, а до того, що час від часу комп’ютерний агент може приймати неадекватні рішення, що буде дратувати гравця, так як це є очевидною «форою».

Також існує більш значна проблема у застосуванні навчання з підкріпленням для прийняття рішень у відеоіграх. У більшості експериментів DeepMind застосовувалися примітивні ігри, як пінг-понг або арканоїд, для яких були отримані високі результати з точки зору ефективності ігрової стратегії. Усі ці ігри об’єднував той факт, що агент відносно часто отримує винагороду, позитивну чи негативну. Це

дозволяло агенту без проблем визначити які дії є оптимальними, а які – ні для різних станів. При цьому постає питання: як буде працювати даний алгоритм, якщо нагорода настільки довгострокова, що для її отримання агент має виконати певну складну послідовність дій. Оскільки при застосуванні машинного навчання з підкріпленням, оптимальні рішення знаходяться шляхом спроб і помилок, системі доведеться намагатися знайти рішення випадковим чином без ніяких проміжних індикаторів успіху, що може виявитися надзвичайно довгим процесом в залежності від складності гри. Згідно до [22] на Atari присутня гра, для якої застосування алгоритму DQN виявилося неефективним, вона називається «Montezuma's Revenge». Ця гра відрізняється від інших перевірених відносною рідкістю винагороди. Агент отримує винагороду тільки після завершення рівню, для чого необхідно виконати конкретну серію дій протягом тривалого періоду часу. У випадку з першою кімнатою «Montezuma's Revenge», яку зображено на рисунку 6.3, це: спуск по сходах, стрибок на канат, стрибок на платформу, спуск по іншій драбині, стрибок через рухомого ворога і підйом по сходам. Усі ці дії має виконати агент лише для того, щоб отримати перший ключ у першій кімнаті.

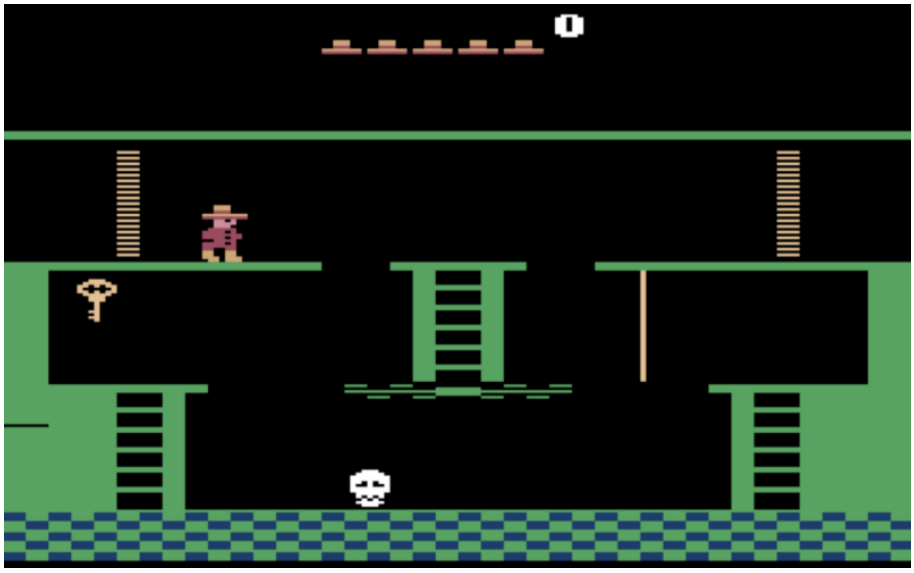


Рисунок 6.3 – Перша кімната у «Montezuma's Revenge»

На першому рівні наявно ще 23 таких кімнати, по яким агент може переміщатися для того, щоб завершити рівень, повну мапу якого зображено на

рисунку 6.4. Гравець може програти багатьма різноманітними способами, в тому числі, впасти великої висоти, після чого йому доведеться починати рівень з самого початку, що робить задачу значно складнішою, знижуючи шанси агента віднайти правильну послідовність дій до того моменту, доки він не програє.

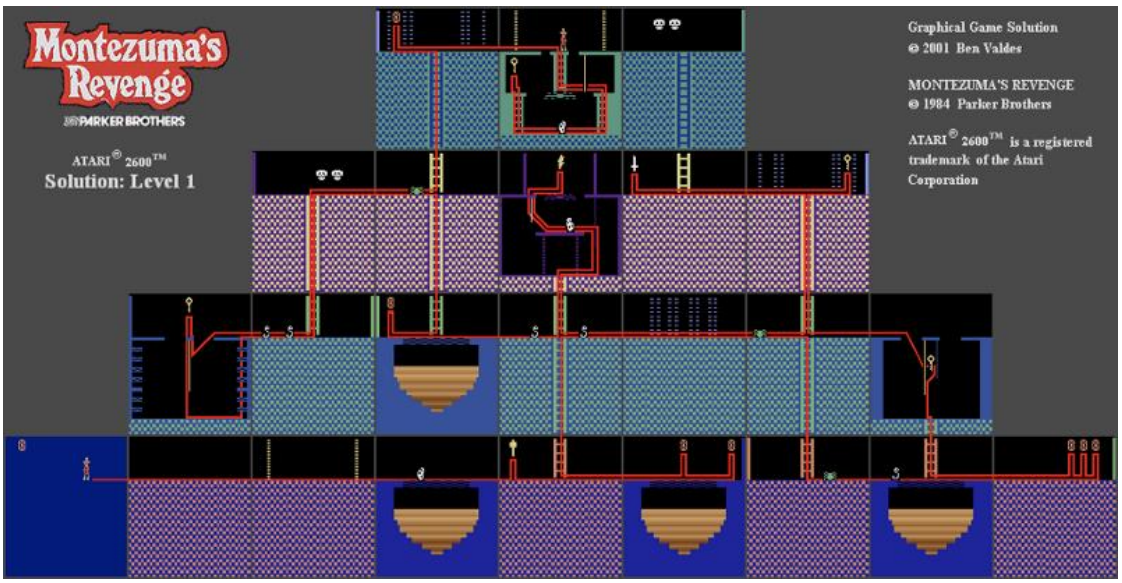


Рисунок 6.4 – Повна мапа першого рівню «Montezuma’s Revenge»

Через складність цієї гри, вона була розглянута як свого роду виклик для дослідників RL-алгоритмів. Навіть на даний момент всі відомі рішення цієї задачі засновані на наданні нейронній мережі тренувальних даних у вигляді відеозаписів прохождення даної гри людьми, що робить ці методи неможливими для використання у контексті даної дисертації. Загалом, використання демонстрацій є ефективним способом надання агентам значних знань про задачі гри та особливості ігрових механік. Живі істоти навчаються аналогічним чином. Ключ до здатності людини вчитися на демонстраціях - це здатність абстрактно сприймати побачене і використати отриманий досвід на численному наборі ситуацій, а не лише у точній копії проблеми, рішення якої було продемонстровано. У випадку з «Montezuma’s Revenge» ефективними виявилися не алгоритми, що намагалися знайти деяке загальне рішення для гри (як у випадку досліджень DeepMind), а засоби експлуатації основної слабкості гри: її постійності. Кожен раз, коли людина або агент грає у дану, вони перед ними завжди постає одна й та сама задача проходження певного

незмінного набору кімнат. Таким чином, простого запам'ятовування рухів по кожній кімнаті достатньо, щоб пройти гру. Хоча це не обов'язково було б значним недоліком, як у випадку якщо агенти навчалися би з нуля, але це стає проблемою, коли справа доходить до застосування тренувальних даних у вигляді демонстрацій проходження гри. Зрештою агенти вчилися не проходити гру, а повторювати попередньо визначений набір дій який приводить до проходження конкретного представленого у грі набору кімнат, що не є повним рішенням задачі, адже при доданні нового рівня у дану гру, алгоритм не зможе з ним впоратися незалежно від тривалості навчання і об’єму тренувальних даних на основі попередніх рівнів.

Якщо показати людині будь-яку кімнату у «Montezuma’s Revenge» і попросити описати що потрібно робити у даній грі для її проходження, ця людина зможе швидко і чітко описати послідовність дій на основі візуальних елементів рівню. Найбільш очевидним проявом цього абстрактного аналізу є розпізнавання живим гравцем ключа як бажаного об'єкта, черепа як чогось небезпечного, чого слід уникати, а сходів – як засобу пересування по кімнаті. Ключі можна використати для відкриття дверей, різні поверхи - для уникання ворогів, таким чином у людини в голові починає з’являтися приблизна стратегія гри. Такі міркування і планування спрацюють не лише на одній певній кімнаті, але й на будь-якому можливому подібному рівні або грі. Саме таке міркування є метою досліджень алгоритмів штучного інтелекту, але використання демонстрацій в детермінованому середовищі повністю позбавляє систему необхідності у цих навичках.

Основна проблема працездатності машинного навчання з підкріпленням полягає саме у встановленні взаємозвязків між ігровими елементами, які можуть бути очевидними для гравця, але невідомими для ШІ, доки він не перевірить їх шляхом спроб і помилок. Ця проблема є складною з точки зору закритої системи, про яку агент не може нічного знати до початку взаємодії з нею, саме з такими системами, в основному, мають справу розробники алгоритмів машинного навчання. У контексті задач даної дисертації ігрове середовище розроблюється разом с алгоритмом ШІ, тому поставлені умови відрізняються від тих, з якими, зазвичай, працюють дослідники. Обізнаність розробників про наявні ігрові механізми можливо

використати при розробці ШІ, створивши деяку початкову базу знань, якою агент зміг би користуватися для того, щоб отримати дані про властивості навколишнього середовища, наприклад: які обєкти є небезпечними, з якими обєктами можливо взаємодіяти, тощо. Наявність цих даних значно спростить процес розробки ШІ, а програмний доступ до змінних середовища виключає необхідність механізму розпізнавання. Головна проблема, за даного підходу, полягає у класифікації ігрових станів, адже Q-навчання потребує деякої Q-функції, якою часто служить таблиця, кількість рядків у якій дорівнює кількості ігрових станів. В складних ігрових середовищах можливих станів може бути забагато для того, щоб будувати таблицю і звертатися до неї на кожній ітерації навчання, тому система повинна або узагальнити певним чином ігрові стани до набору умовних перевірок або мати деякий класифікатор, у ролі якого може виступити нейронна мережа.

Слід також відмітити, що демонстрація адаптивних механізмів гри є однією з важливих задач розробника, оскільки без візуального підкріплення гравцю буде важче зрозуміти що місцеві агенти здатні навчатися. Розуміння гравцем принципу роботи штучного інтелекту має велике значення, оскільки це впливає на рішення які він приймає протягом гри. Гравців дратує непостійність ШІ, коли агенти поводяться себе по-різному у однакових ситуаціях, оскільки це може призвести до провалу стратегії гравця, що заснована на спостереженні за ігровим процесом, тому варто одразу попередити гравця що агенти здатні змінювати свою поведінку з часом, щоб він зміг використати цю інформацію і отримати повніший ігровий досвід. "Black & White робить це за рахунок зворотного звязку з гравцем у вигляді елементів інтерфейсу та дизайну, які дозволяють гравцю не лише побачити що його дії впливають на розвиток агентів, але й чітко розуміти як саме і за яких умов це відбувається, завдяки чому гравець дотримує змогу планувати свої подальші дії та експериментувати з ігровим середовищем.

7 ОСОБЛИВОСТІ ЗАСТОСУВАННЯ Q-НАВЧАННЯ

7.1 Дослідження середовища

Для застосування навчання з підкріпленням, агент повинен мати дані про можливу винагороду, яку він, в перспективі, може отримати за свої рішення. Ці дані він отримує коли приймає рішення на основі поточного стану, в який він потрапляє внаслідок своїх рішень, таким чином, працюючи за принципом зворотного зв’язку, як на схемі на рисунку 7.1.1.

Рисунок 7.1.1 – Зворотний зв'язок агента з ігровим середовищем

Постає питання збору репрезентативних навчальних даних, так як наявність тренувальних даних неможлива згідно до проблеми, яка описує дана дисертація. Для того, щоб агент дізнався, які рішення є оптимальними у різних станах середовища, він має побувати у якнайбільшій можливій кількості з цих станів. На відміну від умов при машинному навчанні зі вчителем, агент з підкріпленням має лише доступ до навколишнього середовища з яким він може взаємодіяти, обираючи рішення із множини A. Як наслідок, виникає циклічна проблема: агент потребує достатнього досвіду, щоб вивчити ефективну політику, але він також потребує ефективної політики, щоб отримати достатній досвід. Результатом існування цієї проблеми стало виникнення класу задач у області машинного навчання з підкріпленням, який відповідає за розроблення методики для балансування агентом процесів дослідження

середовища і його експлуатації. В ідеальному випадку, такий підхід повинен заохочувати вивчення агентом навколишнього середовища до того моменту, коли він отримає достатньо даних, щоб приймати обгрунтовані рішення щодо оптимальних дій в тому чи іншому стані. Існує ряд використовуваних підходів до заохочення агента до дослідження, які використовуються у складі політик прийняття рішень для створення дівергенції у стратегії, яку агент вважає оптимальною і, в перспективі, знаходження кращих рішень у певних ситуаціях.

Жадібний підхід

Всі алгоритми навчання з підкріпленням засновані на максимізації винагороди. Наївним підходом до забезпечення прийняття агентом оптимальних рішень в будь-який момент часу є вибір тієї дії, яка, згідно до очікувань агента має призвести до максимальної винагороди, тому цей метод називається жадібним. Виконання дії, яку агент оцінює як оптимальну на даний момент, є прикладом експлуатації: агент використовує свої поточні знання про систему винагород навколишнього середовища, щоб приймати рішення. Проблема жадного підходу полягає в тому, що вона майже завжди досягає локально оптимального рішення і не отримує максимально можливого результату через недостачу даних про середовище, яка є результатом відсутності прагнення агента до дослідження.

Випадковий підхід

Протилежний підхід, на відміну від жадібного, полягає в тому, щоб завжди виконувати випадкову дію. Такий підхід можливий лише в тих випадках, коли випадкова політика є єдиною можливою оптимальною (Наприклад, система є абсолютно нестатичною і змінює свої правила кожної ітерації, що позбавляє навчання сенсу). Зазвичай, такий підхід застосовують на першій ітерації для заповнення буферу досвіду і його подальшого коригування, але надалі не має особливої практичної цінності.

ε-Жадібний підхід

Поєднання жадібного і випадкового підходів дозволяє отримати одну з найбільш використовуваних стратегій дослідження середовища. У цьому підході агент по більшій мірі обирає ті рішення, які вважає оптимальними, але іноді виконує випадкові дії. Таким чином, агент виконує дії, які не можуть бути оцінені як ідеальні, але можуть, у перспективі, надати нову інформацію. В даному підході коефіцієнт ε - це регульований параметр, який визначає ймовірність прийняття випадкового рішення з множини можливих у даному стані. Завдяки своїй простоті та ефективності, цей підхід став одним з найвикористовуваніших методів для більшості проектів зі застосуванням навчання з підкріпленням, включаючи глибоке навчання з підкріпленням та його варіанти. На початку тренувального процесу величина ε часто прирівнюється великому значенню, щоб заохотити агента до дослідження в умовах малої кількості інформації про середовище. Пізніше значення коефіцієнту спадає до деякої невеликої константи (зазвичай, близько 0,1), припускаючи що агент знає більшу частину необхідної інформації про середовище і може почати використовувати свої знання для максимізації винагороди. Незважаючи на поширеність використання, цей метод далекий від оптимального, оскільки він враховує лише чи є певна дія найкориснішою, чи ні.

Метод Больцмана

Під час дослідження бажано використовувати інформацію, яку зібрав агент у вигляді значень Q-функції. У випадку ε-жадібного підходу агент або обирає краще можливе рішення, або абсолютно не використовує накопичений досвід заради дослідження середовища. Метод Больцмана вирішує цю проблему. Замість того, щоб завжди обирати між оптимальною та випадковою діями, цей підхід передбачає вибір рішення зі зваженими ймовірностями. Для цього застосовується софтмакс-функція над значеннями Q-функцій для усіх можливих дій. У цьому випадку вибір дії, який агент оцінює як оптимальний, є найбільш імовірним (але не гарантованим).

Найбільшою перевагою перед ε–жадібним підходом є те, що інформація про цінність ненайоптимальніших дій також враховується. Якщо для агента доступні 4 дії, то в ε–жадібному підході усі три дії, що були оцінені як неоптимальні, розглядаються однаково, але в методі Больцмана ймовірність їх вибору відповідає значенням Q-функції для даних дій у поточному стані. Таким чином, агент може ігнорувати дії, які він оцінює як строго неоптимальні, і приділяти більше уваги потенційно перспективним, але не обов'язково ідеальним рішенням. Основним недоліком даного методу є припущення, яке полягає в тому, що застосування софтмакс-функції над значеннями Q-функції демонструє наскільки агент впевнений у тому, що певне рішення є оптимальним. Насправді це не вірно. Натомість ймовірності вибору певного рішення агентом є мірою того, наскільки оптимальним певне рішення вважає агент, а не наскільки він впевнений у тому, що це рішення – оптимальне. Хоча цей метод демонструє ефективність, він не дозволяє досліджувати середовище у тій мірі ефективності, на яку розраховував автор.

7.2 Варіації алгоритму

Глибоке Q-навчання

При даному підході у якості Q-функції використовується глибока згорткова нейронна мережа з шарами згорткових фільтрів для імітації ефекту рецептивних полів. Навчання з підкріпленням може проходити нестабільно або дівергентно, якщо для представлення Q використовується така нелінійна функція апроксимації, як нейронна мережа. Нестабільність виникає через: кореляції, що присутні у послідовності спостережень, зміни у Q-функції, що можуть істотно змінити політику прийняття рішення і розподіл даних, а також співвідношення між Q і цільовими значеннями. Технологія застосовує методику відтворення досвіду (experience replay), механізм натхненний біологічними процесами, який для навчання використовує випадкову вибірку попередніх прийнятих рішень замість останніх. [23] Це усуває кореляції в послідовності спостережень і згладжує зміни в розподілі даних. Ітераційне

оновлення налаштовує Q на цільові значення, які лише періодично оновлюються, що ще більше знижує кореляцію з ціллю.

Подвійне Q-навчання

Оскільки майбутнє максимальне апроксимоване значення дії в Q-функції оцінюється використанням тієї ж самої Q-функції, що і в поточній політиці прийняття рішень, в зашумлених середовищах процес Q-навчання іноді може призводити до переоцінки значення якості дій, сповільнюючи навчання і зменшуючи ефективність алгоритму. Для рішення даної проблеми було запропоновано методику під назвою подвійне Q-навчання (Double Q-learning)[24]. Подвійне Q-навчання є алгоритмом навчання з підкріпленням з пасивною політикою, за якої для оцінки значення функції використовується інша політика, ніж та, що використовуватиметься для вибору наступної дії. На практиці використовуються дві окремі функції оцінки, що навчаються взаємно симетричним чином, використовуючи окремий досвід: Q^A і Q^B. За подвійного Q-навчання значення відповідних Q-функцій оновлюються за наступними формулами:

$$Q_{t+1}^A(s_t, a_t) = Q_t^A(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q_t^B\left(s_{t+1} \arg\max_a Q_t^A(s_{t+1}, a)\right) - Q_t^A(s_t, a_t)\right)$$

$$Q_{t+1}^B(s_t, a_t) = Q_t^B(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q_t^A\left(s_{t+1} \arg\max_a Q_t^B(s_{t+1}, a)\right) - Q_t^B(s_t, a_t)\right)$$

Таким чином, значення дисконтованого майбутнього рішення оцінюється за допомогою іншої політики аніж для вибору дії у поточному стані, що, в результаті, вирішує питання надлишкової переоцінки. Згодом, даний алгоритм було поєднано з глибоким навчанням (як у DQN-підході), результатом чого став комбіновий підхід під назвою подвіний DQN, що, згідно до досліджень [24] перевершує звичаний алгоритм DQN.

8 ТЕСТУВАННЯ АГЕНТІВ У КОМПЕТЕТИВНОМУ СЕРЕДОВИЩІ

8.1 Розробка тестового середовища

Згідно до [25], некооперативна гра уявляє з себе систему, де декілька сторін ведуть боротьбу за реалізацію своїх інтересів. Кожен із гравців приймає рішення з певної наданою системою множини рішень, які згодом ведуть до перемоги або програшу. Виграш є головним показником ефективності стратегії, таким чином середовище повинно повідомляти агентів про їх результати для того, щоб вони отримали дані для подальшого надання переваги певним рішенням перед іншими. При цьому, правила гри мають бути достатньо гнучкими для перевірки збереження ефективності алгоритмів агентів за зміни умов. Враховуючи вимоги до експерименту, було обрано гру на основі дилеми в'язня[26] - гри з ненульовою сумою, в якій гравці прагнуть отримати вигоду, співпрацюючи або зраджуючи один одного.

Правила гри:

– Граючи з опонентом, агент повинен прийняти рішення співпрацювати або зрадити його, не знаючи що обрав інший. В залежності від обраних рішень, обидва агенти отримують бали згідно до таблиці 1.

– В грі беруть участь N агентів і кожен має зіграти з кожним, таким чином кількість ігор в одному турі складає $N!$. При цьому, агенти однакового типу також грають один з одним.

– Кожна гра складається з r раундів, обидва гравця дізнаються про результати свого вибору після кожного раунду.

– Після кожного туру n гірших агентів за кількістю набраних балів видаляються з гри, а n кращих копіюються. Гра продовжується доки не буде відіграно T турів, або доки не залишиться лише один тип агентів.

Бали, що нараховуються агентам, в залежності від результату гри вказано у таблиці 7.1

Таблиця 7.1 – Правила дилеми в’язня

Результати агенту	Опонент довірився	Опонент зрадив
Агент довірився	+2 бали	-1 бал
Агент зрадив	+3 бали	0 балів

Згідно до [26], оптимальна стратегія може радикально змінюватися в залежності від зміни винагороди за різні дії агента, стратегій супротивників та співвідношення стратегій задіяних у грі. Оптимальною стратегією за наведених умов вважається «Око за око», за якої агент спочатку обирає довіритися, а потім повторює останній хід партнера. Цей алгоритм буде використано у якості контрольного разом із алгоритмами «Завжди довіряти» та «Завжди зраджувати».

Згідно до поставленої задачі, алгоритм агентів не матиме тренувальних даних і зобов’язаний приймати рішення шляхом спроб і помилок, експлуатуючи зворотній зв'язок із своїм середовищем. Для вирішення цієї задачі буде застосовано машинне навчання з підкріпленням, агенти намагатимуться отримати якнайкращий результат. У ході навчання алгоритм ітеративно обиратиме рішення з найбільшим значенням $Q(s,a)$ та застосовуватиме рівняння Беллмана для оновлення значень Q-таблиці після отримання винагороди R за обране рішення, у вигляді:

$$Q(s,a) = Q(s,a) + \alpha[R + \gamma * maxQ(s',a') - Q(s,a)]$$

Агент прийматиме рішення згідно до методу Больцмана. При цьому, було вирішено виставити обмеження максимально допустимої ймовірності вибору певного рішення до $p \leq 0.95$, таким чином система завжди матиме шанс обрати рішення з не найвищим $Q(s,a)$ з двох альтернатив з мінімальною ймовірністю:

$$p'_{min} = 1 - p_{max} = 0.05.$$

Алгоритм роботи агенту наведено у додатку А. Для повноти експерименту було вирішено задіяти декілька адаптивних алгоритмів у компететивному середовищі. Постає проблема: в більшості досліджень алгоритмів машинного навчання, зазвичай, відомі вхід і бажаний вихід, а на алгоритм покладається задача генерування функції, яка повертатиме очікуваний результат в залежності від вхідних параметрів. Кажучи інакше: створення «чорного ящику» на основі тренувальних даних. У даній задачі «чорний ящик» вже наявний і представлений середовищем, яке можна представити як функцію залежності результуючої винагороди і стану від дій агентів, таким чином проблема зводиться до створення коректних вхідних параметрів для функції, що не піддається диференціації, що виключає класичні засоби регулювання через зворотний зв'язок. Згідно до [27], задачі максимізації функції недиференційованої у точці екстремуму відповідають задачам негладкої оптимізації і не можуть бути вирішені безпосередньо через відсутність можливості знайти градієнт в будь-якій точці (в першу чергу – в точці екстремуму). [28] підмічає що в такому випадку можливо застосувати метод без вираховування похідної, наприклад, генетичний алгоритм. Ефективність генетичного алгоритму зменшується по мірі збільшення виміру задачі [29], але у динамічному середовищі це не є проблемою, оскільки точка екстремуму постійно змінюється, що може означати, що система в будь-якому разі ніколи не знайде оптимальне рішення незалежно від застосовуваного алгоритму, що робить швидкість наближення до оптимуму важливішою аніж точність рішення. [30] демонструє вдале застосування генетичного алгоритму у відеогрі, відмічаючи що метою штучного інтелекту є не знайдення ідеальної непереможної стратегії, а наближення до навичок гравця. Для даного експерименту було вирішено об’єднати нейронну мережу як функцію залежності рішення агента від поточного стану гри разом з генетичним алгоритмом як методом тренування мережі. Мережа матиме архітектуру, вказану на рисунку 8.1.1. На рисунку 8.1.1 a – це кількість можливих результатів одного раунду, в даному випадку $a = 4,\ r$ – кількість раундів однієї гри. Вхідний вектор описує поточний стан гри і має: $(r-1)a$ нейронів, оскільки в першому раунді агент не має вихідних даних для прийняття рішення. З цієї самої причини прихований шар мережі має нейрон зсуву B_1, який впливатиме на

рішення агенту в першому раунді. Для мережі було обрано сигмоїдну функцію активації.

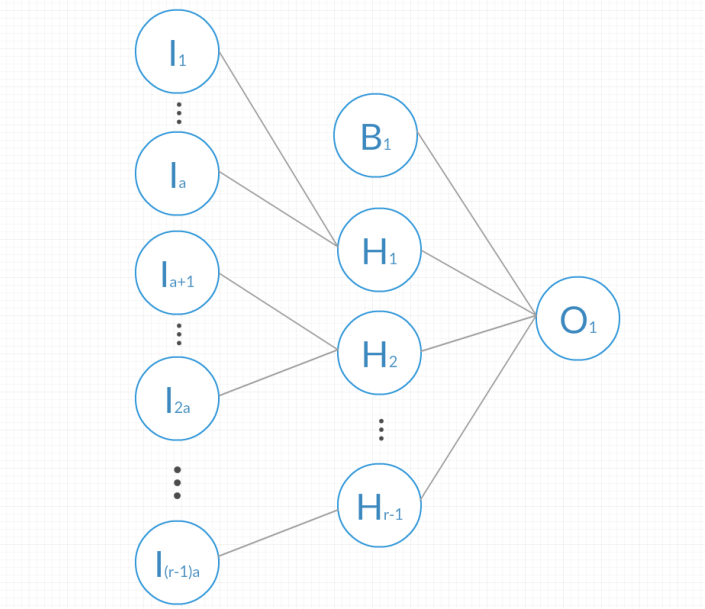


Рисунок 8.1.1 – Схема нейронної мережі для прийняття рішення у дилемі в’язня

Геномом генетичного алгоритму служитиме вектор вагів нейронної мережі. В кінці туру між геномами кращої половини агентів за результатами гри буде виконано кросинговер у випадкових точках перетину з мутацією одного випадкового гену для отримання наступного покоління, що міститиме властивості кращих представників попереднього. Оскільки бажані властивості мережі можуть радикально змінюватися у динамічному середовищі, важливим є притік у популяцію нових ознак, таким чином, гірша половина агентів буде замінятися на цілком випадково іцініалізовану групу. Алгоритм роботу агенту наведено у додатку Б.

8.2 Результати контрольних ігрових сесій

Було розроблено програму, що дозволяє проводити змагання між довільною кількістю агентів різного типу за наведеними правилами гри. При цьому у ігрових правилах можлива зміна таких параметрів, як: кількість раундів однієї гри, кількість

балів, які додаються/знімаються в усіх можливих ігрових ситуаціях, кількість агентів що виключаються кожного туру та кількість клонів кращого агенту кожного туру. Різні стратегії можуть виявитися по-різному ефективні в залежності від правил гри, тому можливість випадкової генерації властивостей ігрового середовища дозволить отримати повнішу картину щодо ефективності адаптивних алгоритмів. У якості першого контрольного туру було виконано розрахунок результатів гри між агентами зі статичними стратегіями: «Око за око» (Агент типу I), «Завжди довірятися» (Агент типу N), «Завжди зраджувати» (Агент типу L). Контрольне змагання містило 10 агентів кожного типу та проходило протягом 20 турів із заміною одного гравця у кожному турі.

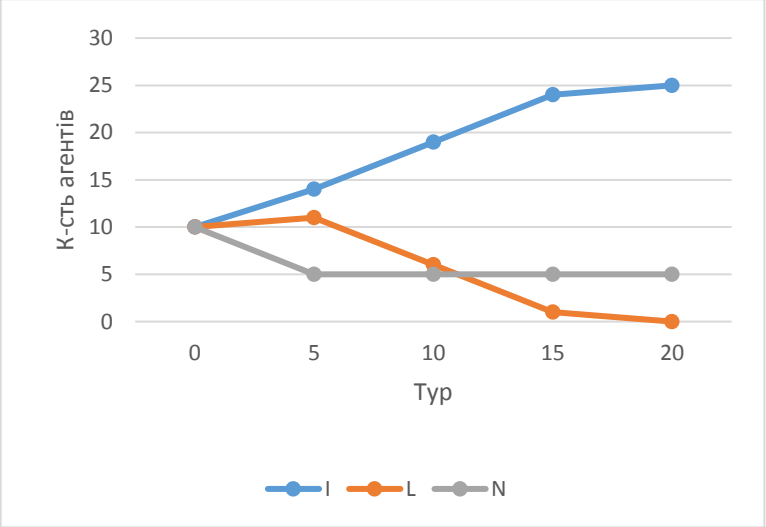


Рис. 8.2.1 – Результати збалансованого контрольного ігрового турніру

Як видно на графіку на рисунку 8.2.1 з самого початку гри, тип I, як і очікувалося, проявив себе як агент з локально-оптимальною стратегією, залишившись переможцем у кінці як за кількістю агентів, так і за середнім балом. Тип L зміг поширити популяцію лише коли в грі було достатньо агентів типу N, після чого був винищений за відсутності умов отримувати достатньо балів за рахунок інших агентів, в той час, як тип N і тип I здатні на співіснування. У другому контрольному змаганні було змінено співвідношення розмірів стартових груп. Для демонстрації повної картини було зіграно 40 турів замість 20. Результати представлено на рисунку 8.2.2.

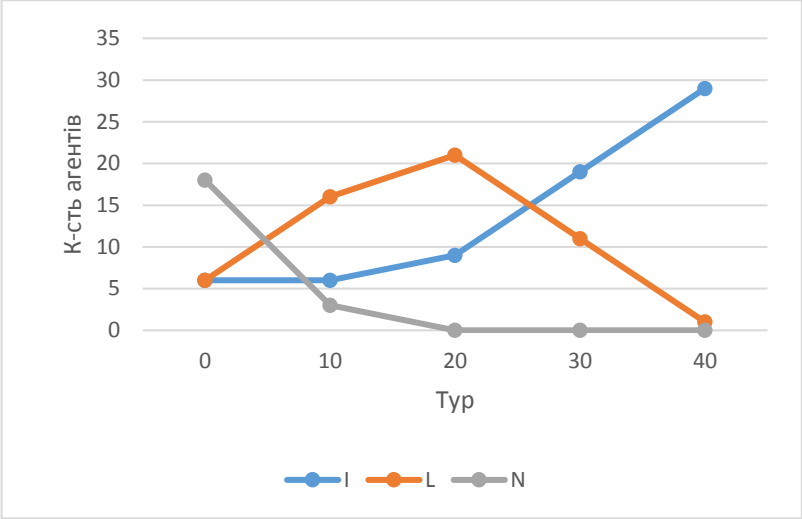


Рис. 8.2.2 – Результати незбалансованого контрольного ігрового турніру

З більшою стартовою популяцію агентів N, стратегія агентів L надала їм значну перевагу до 18-го туру, після чого агентів N не залишилося, таким чином залишаючи групу L переможцями у 20-му турі. Надалі здатні співіснувати один з одним агенти I отримали перевагу над агентами L, втратившими можливість отримувати бали. Таким чином, можна зробити висновок, що в залежності від початкового стану середовища, різні стратегії можуть виявитися ефективними на різних етапах гри. При створенні середовища перед розробником постає прогнозування можливих ігрових сценаріїв та задання поведінки штучного інтелекту для якнайбільшої кількості з них, але чим складніше середовище, тим важче передбачити усі можливі ходи подій, що часто призводить до неадекватної поведінки комп'ютерних агентів [31]. Саме цю проблему мають вирішити адаптивні агенти.

8.3 Результати ігровий сесій з RL-агентами

Для експерименту з агентами на основі навчання з підкріпленням вибірку було збільшено до 100 агентів кожного типу, а протяжність гри збільшено до 400 турів. Для агентів RL були обрані налаштування: $\alpha = 0.9$, $\gamma = 0.5$

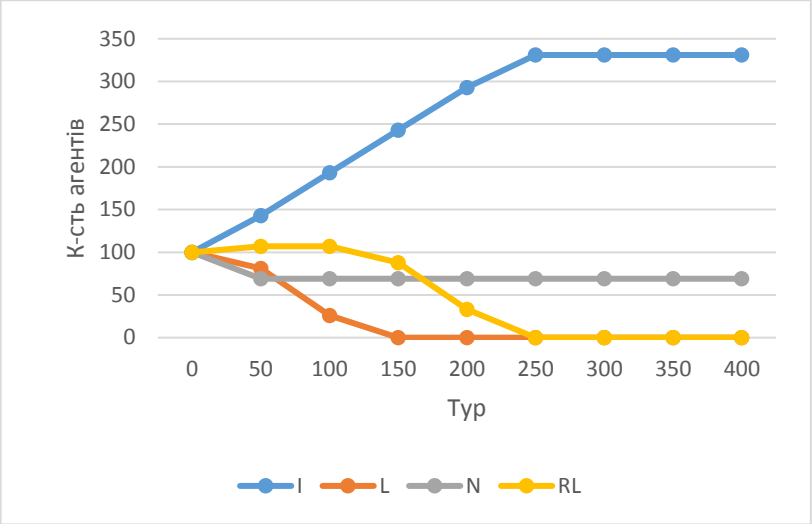


Рисунок 8.3.1 – Результати турніру з RL-агентами з оновленням Q-таблиці після кожного раунду

Як видно на рисунку 8.3.1, агенти RL виявилися неефективними, зникнувши із середовища і залишивши типи I та N подібно до першого контрольного тесту. Це пояснюється тим, що кожен агент отримує бали після кожного раунду. Обираючи рішення зрадити на першому кроці, агент RL отримував, в середньому, більше балів, ніж обираючи довіритися, що призводило до зростання ймовірності обрати зраду на першому кроці у майбутньому. Це, в свою, чергу, призводить до зради агентами I у другому раунді і втрати балів у довгостроковій перспективі. Було прийнято рішення оновлювати Q-таблицю після закінчення ігрової сесії, перераховуючи значення для усіх обраних рішень згідно до кумулятивної винагороди. В свою чергу, це може призвести до зменшення ймовірності повторення бажаної стратегії в тому випадку, коли помилку було припущено лише на останньому кроці. Ця проблема називається «проблемою призначення заслуги»[32]. Згідно до закону ефекту [33] закріпленню підлягають лише ті рішення в певній ситуації, які приносять бажаний ефект, таким чином, статистично, правильні рішення на початкових стадіях гри будуть призводити до отримання більшої кількості балів і у довгостроковій перспективі, аніж невдалі рішення, тому слід очікувати що протягом достатньої кількості ітерацій сприятливі обставини мають виникати частіше по мірі навчання, що означає, потенційно, більшу кількість результуючих балів.

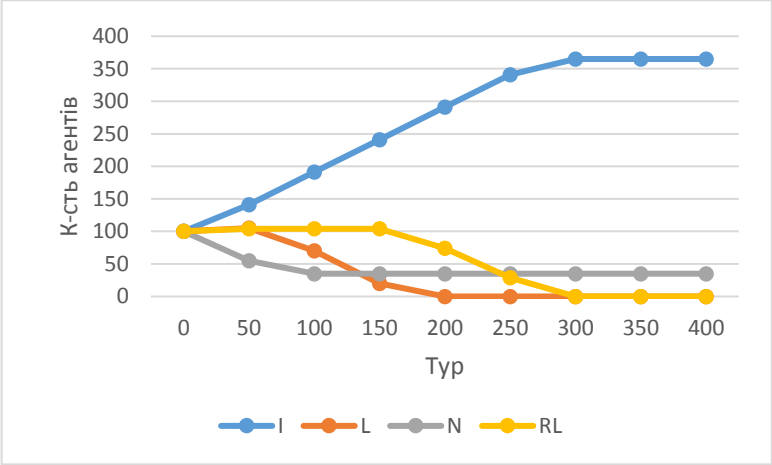


Рисунок 8.3.2 – Результати турніру з RL-агентами з оновленням Q-таблиці в кінці ігрової сесії

Як видно з графіку на рисунку 8.3.2, зміна алгоритму начислення винагороди призвела до збільшення середнього часу життя популяції агентів типу RL на 13%. Дослідження логів програми показало, що зникнення популяції пов’язано з неправильним значенням винагороди, що присвоюється в залежності від кінця ігрової сесії. Згідно до правил гри, агент RL може отримати результат $R = 3$, граючи проти агента I за стратегією агента L. Оскільки цей результат більше нуля, він збільшує ймовірність повторення такої поведінки у майбутньому, згідно до алгоритму Q-навчання, але є провальним з точки зору гри.

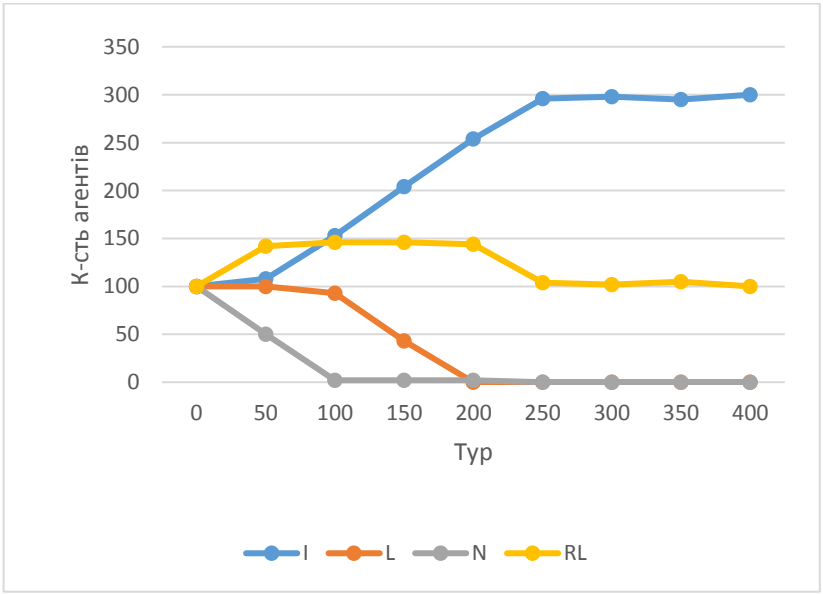


Рисунок 8.3.4 – Результати турніру з RL-агентами з застосуванням відносних винагород

Було прийнято рішення корегувати винагороду для оновлення Q-таблиці відносно середнього значення винагород усіх RL-агентів, таким чином заохочуючи лише ту поведінку, яка призводить до результатів вище середнього. Як видно на рисунку 8.3.4, кількість агентів RL залишається стабільною до 170-го раунду, як у випадку без нормалізації винагороди. Зі зменшенням кількості агентів N і L, агенти RL також починають зникати, але популяція не вимирає повністю, залишаючись співіснувати з агентами I у пропорціях 3:1. Згідно до логів програми, агенти RL, маючи доступ до колективної Q-таблиці, змогли віднайти ефективну стратегію на 5-му турі, яка є копією стратегії агентів I за виключенням схиляння зо зради на останньому турі через можливість отримати більшу кількість балів таким чином. В подальшому зменшення популяції RL-агентів пояснюється відхиляннням від бажаних рішень у кожному раунді з ймовірністю $\varepsilon = 0.05$, таким чином, ймовірність закінчення гри без відхилення від локально оптимальної стратегії складає:

$$P = 0.95^{r=5} = 0.773,$$

і буде зменшуватися по мірі збільшення кількості раундів гри. Вирішити цю проблему можливо лінійним зменшенням коефіцієнту ε з ітераціями навчання, але це зробить агента менш спроможним до дослідження середовища, яке може змінюватися згідно до поставлених задач. Таким чином, показником того, що результати навчання перестають бути валідними стане падіння середньої кількості балів, що отримають агенти. Це означає, що можливо створити деякі нижній та верхній пороги ефективності за досягнення яких коефіцієнт ε почне змінювати своє значення, стимулюючи скидання або закріплення поведінки відповідно. Нехай є деякий коефіцієнт $l \in (0,1)$, який виступає модифікатором ε в залежності від успішності результатів агенту, так, що:

$$\varepsilon' = \begin{cases} \varepsilon * (1 + l), & x \leq 0 \\ \varepsilon * (1 - l), & x > 0 \end{cases}$$

де x – деякий коефіцієнт, що приймає додатне значення коли агент отримує бажаний результат у грі,

ε' - значення коефіцієнту ε на наступній ітерації навчання

Алгоритм було модифіковано з урахуванням коефіцієнту l без створення залежності між значенням ε та відхилення отриманого агентом результату від попереднього. Також, ε завжди має знаходитися на проміжку [0.95, 1] для попередження ситуації, коли завелике значення помилки не дозволить системі експлуатувати накопичений досвід для покращення результату. Для оновленого алгоритму було перезапущено турнір, результати якого наведені на рисунку 8.3.5.

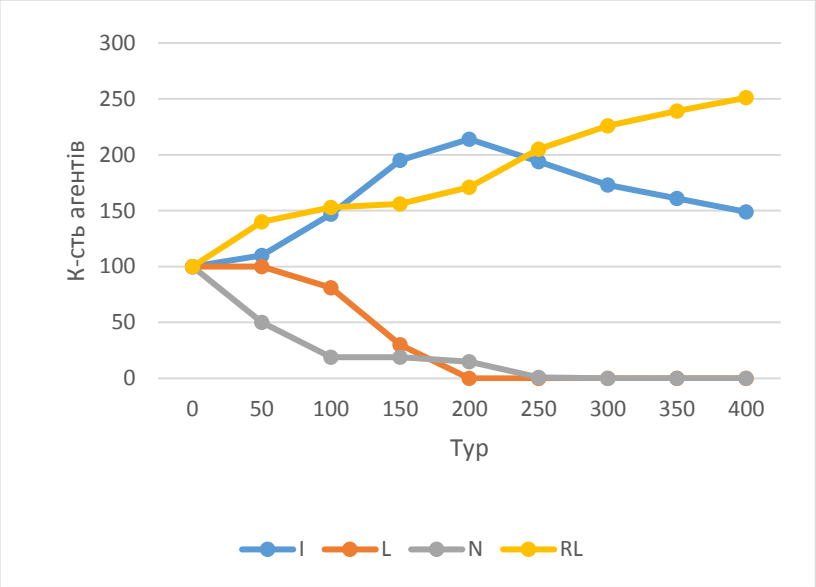


Рисунок 8.3.5 – Результати турніру з RL-агентами з застосуванням плаваючого ε

З плаваючим ε-коефіцієнтом RL-агентам вдалося перемогти I-агентів у кінцевому результаті. Слід відмітити, що існує тимчасовий проміжок, де RL-агенти потрапили на другу позицію, цей період починається з моменту коли кількість N-агентів стала занадто низькою щоб експлуатувати їх, що призвело до різкого падіння ефективності стратегій, в яких переважає зрада. Різниця між L-агентами і RL-агентами в тому, що перші зникли коли надмірна зрада почала призводити до втрати балів у середньому, а другі – пристосувалися до нового середовища, почавши експлуатувати слабкість I-агентів, що мали високу популяцію.

8.4 Результати ігровий сесій з генетичними агентами

Для експерименту було взято 100 агентів на основі генетичного алгоритму (агенти типу G) та 100 агентів кожного контрольного типу.

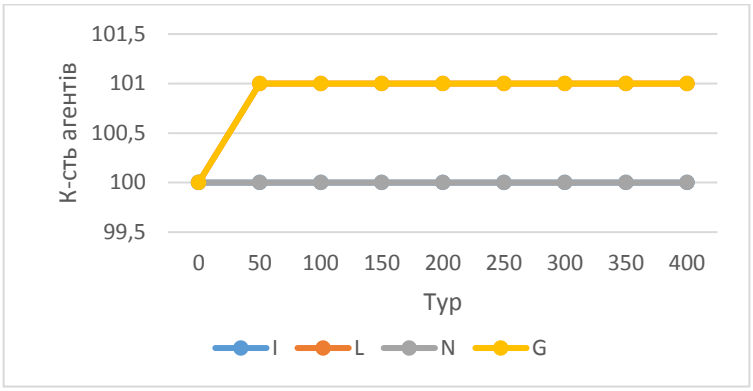


Рис. 8.4.1 – Результати збалансованого турніру з G-агентами

Тест показав, що у 399 із 400 турів найкращий та найгірший агенти за результатом належали до групи агентів на основі генетичного алгоритму. Графік результатів турніру наведено на рисунку 7.4.1. Гірші результати з'являлися через генерацію у кожному поколінні випадкових геномів з метою внесення нових ознак до популяції. Результати перевірки ефективності алгоритму в умовах обмеженої початкової популяції продемонстровано на рисунку 8.4.2.

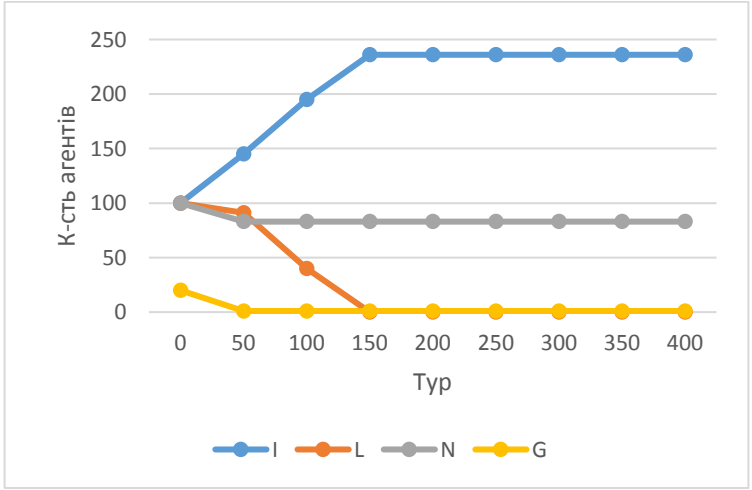


Рисунок 8.4.2 – Результати незбалансованого турніру з G-агентами

В умовах обмеженої популяції у розмірі 30-и агентів алгоритм проявив себе неефективно. Після 38-го туру залишився 1 агент типу G який залишався у вибірці гравців до 400-го туру включно. Без джерела нових генетичних ознак та в умовах постійного зменшення групи алгоритму було достатньо щоб виробити локально оптимальну стратегію, але не залишити достатньої вибірки на випадок зміни параметрів середовища, яка призведе до зникання останньої одиниці групи G.

8.5 Результати комбінованих ігрових сесій

Для повного статичного тесту було взято по 100 агентів наступних груп: трьох контрольних, агентів на основі навчання з підкріпленням, агентів на основі генетичного алгоритму.

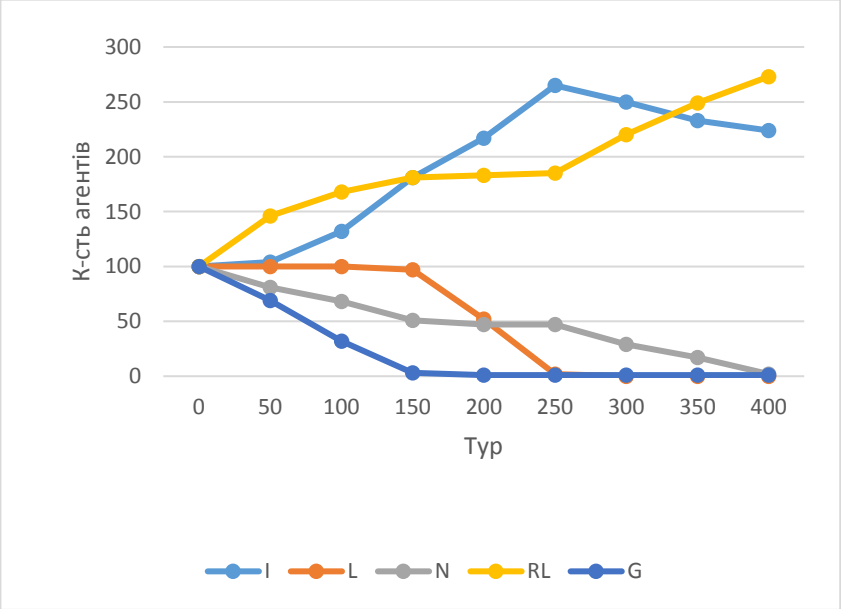


Рисунок 8.5.1 – Результати комбінованого турніру

Як видно на рисунку 8.5.1, загальний розподіл відповідає двом фінальним окремим тестам з агентами на основі машинного навчання. Варто відмітити, що 1 агент типу G залишається у виборці до кінця гри. Таким чином, можна зробити висновки, що без попереднього навчання динамічні адаптивні алгоритми ефективніші у короткостроковій перспективі до тих пір, доки кумулятивна помилка за рахунок

дослідження середовища не перевищить винагороду статичної експлуатації. У таблиці 2 наведено результати серії експериментів з випадковими групами агентів. Було проведено 5000 турнірів, кожні 50 турнірів випадково змінювалися параметри гри: кількість раундів та винагороди за усі можливі результати раунду (від 1 до 7 по модулю). Також із 5 груп обиралося випадковим чином 4, що будуть брати участь в наступних 50-и турах. При цьому, з метою збереження ігрової логіки, винагорода за успішну зраду завжди перевищувала винагороду за довіру, а штраф за невдалу довіру завжди був найгіршим показником. Кожні 50 турнірів обиралося 2 переможці за показниками: результуюча кількість агентів та результуючий середній бал.

Таблиця 8.5.1 – Результати 100 серій по 50 турнірів

	RL	G	I	L	N
Перемог по к-сті	53	33	0	12	2
Перемог по середн. балу	44	37	0	12	7
Програшів по к-сті	0	14	0	84	2
Програшів по середн. балу	5	16	66	4	9

Таблиця 8.5.1 демонструє, що в умовах постійних змін параметрів середовища адаптивні алгоритми демонструють більшу ефективність, аніж статичні. Слід також помітити, що агенти типу I, будучи ефективними при правилах за замовчуванням, не отримали жодної перемоги в умовах динамічного середовища, що говорить про високу залежність оптимальної стратегії від початкових умов. Агенти на основі генетичного алгоритму демонструють широкий діапазон результатів, що пов’язано з алгоритмом пошуку оптимальних рішень, згідно до якого, частина агентів кожного покоління генерує випадкові ваги нейронної мережі з метою безперервного дослідження середовища, що, згодом призводить до появи малоефективних представників покоління. При цьому, агенти типу G отримали перемогу у 35% випадків, що є задовільним результатом, враховуючи, що агенти типу RL перемагали у 43.5% випадків, таким чином надаючи комбінованій групі без попереднього навчання в умовах агресивного середовища 78.5% перемог від усіх турнірів.

Результати серії тестів продемонстрували задовільний рівень ефективності агентів на основі машинного навчання без можливості попереднього тренування. Слід виокремити основні фактори, що впливають на ефективність агентів при імплементації адаптивних алгоритмів у ігровій системі.

При застосуванні генетичних алгоритмів важливо створити достатню вибірку, у якій частина агентів зможе генерувати нові ознаки для дослідження змінного середовища. В умовах відносно малої стартової популяції агенти не здатні забезпечити достатній притік нових генів для рішення задачі пошуку локально-оптимальної стратегії. Рекомендується застосовувати підхід у іграх жанру «стратегія», так як вони часто мають велику кількість ігрових одиниць – «юнітів», на яких можливо покласти задачі задачі оптимізації геному. До того ж, на відміну від проведеного експерименту, в ігровій системі, зазвичай, є можливість створити штучний притік агентів в умовах необхідності (як у випадку, коли в ході експерименту залишився 1 агент типу G, не здатний передати свої ознаки). Слід, також зазначити, що при існуванні в ігровому середовищі відносно великої кількості агентів, що керуються комп'ютером, гравцю буде складніше помітити локально-неоптимальні рішення, що приймають окремі агенти, досліджуючи середовище, що потенційно може значно нівелювати проблему існування слабких представників у популяції.

Агенти на основі навчання з підкріпленням демонструють значну ефективність у пошуку оптимальних стратегій в умовах змінного середовища. Оскільки, на відміну від генетичного підходу, агенти типу RL мали колективну Q-таблицю, вони могли ефективно навчатися на помилках своїх представників та обмінюватися результатами дослідження середовища, завдяки чому розмір групи агентів не має настільки життєво-важливо значення як у випадку з генетичними алгоритмами, що збільшує відносну область застосування RL-агентів, так як не у всіх іграх є можливість генерувати значні популяції агентів для надання групі можливості створювати достатньо ознак для збереження життєздатності. Тести показали що застосування статичного ε-коефіцієнту призводить до потенційної втрати переваги на етапі вироблення ефективної стратегії, але не може бути виключено через

необхідність підтримки тенденції до навчання. Оскільки даний метод не потребує популяції незалежних агентів, його можна використовувати у механізмах глобального контролю ігрової системи, наприклад, для балансування складності [34] та підтримки рівня зацікавленості у гравця [35] шляхом маніпуляції параметрами середовища з метою створення унікальних ігрових сценаріїв.

Для тренування системи необхідно визначити коректну систему винагород, яка може не співпадати з системою винагород з точки зору гравця, так як з метою заохочення гравця, ігри часто видають позитивні бали за невдалі результати. Рекомендується використовувати відносну систему винагород, приховану від гравця, таким чином розробник може обрати пріоритети поведінки агенту в залежності від бажаних значень параметрів ігрового процесу. Наприклад, можливо створити систему винагород за утворення певних ігрових сценаріїв: позбавлення гравця певних ресурсів, провокації його до періодичних змін у стратегії, зміни рівня складності в залежності від його навичок та досягнення певних значень інших цільових параметрів середовища, які можуть реєструватися під час гри.

У експерименті не розглядалося глибоке навчання з підкріпленням через низьку ймовірність наявності достатньої кількості даних для навчання під час ігровий сесій на клієнтському пристрої, але цю техніку можливо використати у онлайн-іграх зі збором даних з кластеру клієнтів та тренування мережі на ігрових серверах для зняття навантаження з клієнтського пристрою. Таким чином можливо виконувати балансування асиметричних ігрових механік та тренування ігрових ботів на основі поведінки гравців. Для виконаного тесту було застосовано Q-таблицю, яка стає неефективним рішенням з точки зору часу навчання по мірі збільшення кількості ігрових станів. Натомість, можливо, у якості стану використовувати вектор значень, що представляє собою набір обраних параметрів, згідно до яких має виконуватися адаптація агентів. При цьому, розробник зможе сам обрати від яких параметрів буде залежати адаптивний механізм, таким чином не лише обмежуючи обчислювальне навантаження, але й чітко контролюючи властивості самонавчального ігрового механізму, необхідність у чому є обов’язковою при розробленні масового продукту.

9 ІМПЛЕМЕНТАЦІЯ НАВЧАННЯ З ПІДКРІПЛЕННЯМ У ІГРОВОМУ ШІ

9.1 Розробка шаблону проектування

На основі отриманих результатів експерименту можливо стверджувати, що навчання з підкріпленням виявилося ефективним методом для реалізації адаптивного механізму у ігровій системі. Слід відмітити, що у ході експерименту деякі рішення щодо принципу роботи системи було прийнято виходячи з особливостей розглянутого ігрового середовища, тому у даному розділі методологія імплементації буде описана у загальному вигляді, спираючись на аналіз існуючих рішень, результати експерименту та правила ігрового дизайну [36]. У сфері розробки програмного забезпечення деяка повторювана архітектурна конструкція, що уявляє собою рішення певної загальної проблеми проектування називається шаблоном проектування [37]. В даному випадку проблемою проектування є використання навчання з підкріпленням у відеоігровому ШІ для отримання стійкого адаптивного ігрового механізму з властивостями згідно до потреб розробника, тому надалі описувана методологія імплементації RL у ігрових системах буде визначатися як шаблон проектування «Q-агент».

Шаблон «RL-агент» повинен визначати зв'язок агентів з ігровим середовищем, так як його реалізація необхідна для обміну даними про дії та нагороди у системі навчання з підкріпленням. Кожен агент має посилатися на деякий модуль прийняття рішень, бути здатний повідомляти його про свої останні рішення, та отриману за них винагороду для того, щоб модуль мав змогу виконувати навчання згідно до обраної розробником стратегії (Q-learning, DQN, і.т.д.). При цьому варто відмітити, що, агенти можуть існувати кластером і є сенс використовувати їх сумарний досвід для швидшого і ефективнішого навчання з підкріпленням, за умови що вони існують і діють в одному середовищі. Таким чином, одному модулю прийняття рішень може відповідати декілька агентів, в той час як кожен агент має посилатися лише на один такий модуль. Припустимо, що розробник вирішує створити деякий клас-

ініціалізатор для генерації нових агентів, який іцініалізує їх та додає в ігрове середовище, закріпивши за певним RL-модулем. Для цього можливо використати шаблон «Фабричний метод», структуру якого вказано на рисунку 9.1.1 [38].

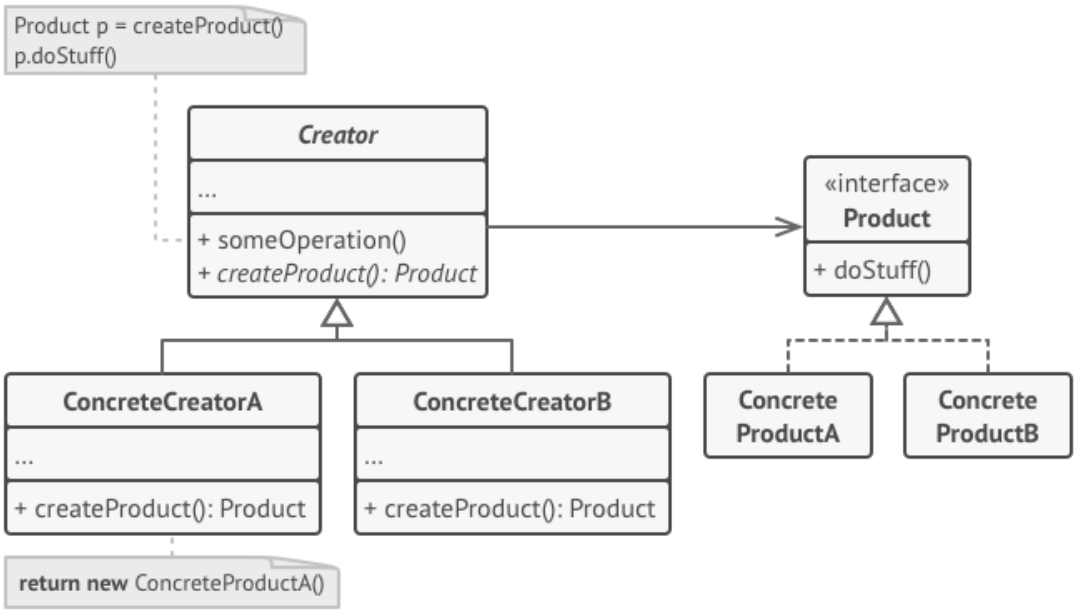


Рисунок 9.1.1 – Структура шаблону «фабричний метод»

За даного шаблону існує деякий клас «Creator», від якого наслідуються так звані «фабрики», що здатні породжувати об’єкти певних типів. При застосуванні приведеного підходу у контексті визначеної задачі постає питання засобу введення деякого модулю прийняття рішень, посилання на який мають зберігати окремі агенти, породжені фабриками. Припустимо, для кожного типу агенту застосовується окремий RL-модуль. В такому випадку логічно зберігати даний об’єкт у відповідному класі агента у вигляді статичного поля, тоді це гарантує наявність у кожного кластеру агентів власного RL-модулю. Тоді виникає проблема: у випадку якщо деякий клас AgentB наслідується від AgentA, він не зможе перевизначити даний модуль, оскільки неможливо перевантажити статичне поле, це призведе до того, що обєкти AgentB будуть використовувати движок рішень створений для об’єктів AgentA. Таким чином даний підхід не є вірним. Натомість, припустимо, що посилання на RL-модуль вказується безпосередньо у екземплярі агента кожного типу. Це можливо, оскільки

кожен тип агентів ініціалізується окремою фабрикою незалежно від того, як вони пов’язані між собою. Тоді постає питання ініціалізації самих RL-модулів, оскільки їх конфігурація залежить від доступного агентам простору дій. У кожного типу агенту має бути певний набір можливих дій, визначених розробником, які є частиною марківського процесу рішень, що представляється RL-модулем. Припустимо, існує деякий інтерфейс IAction, який визначає функціональні можливості обєкту-дії через імплементацію методу деякого Execute() аналогічно до шаблону проектування команда, який вказано на рисунку 9.1.2 [39].

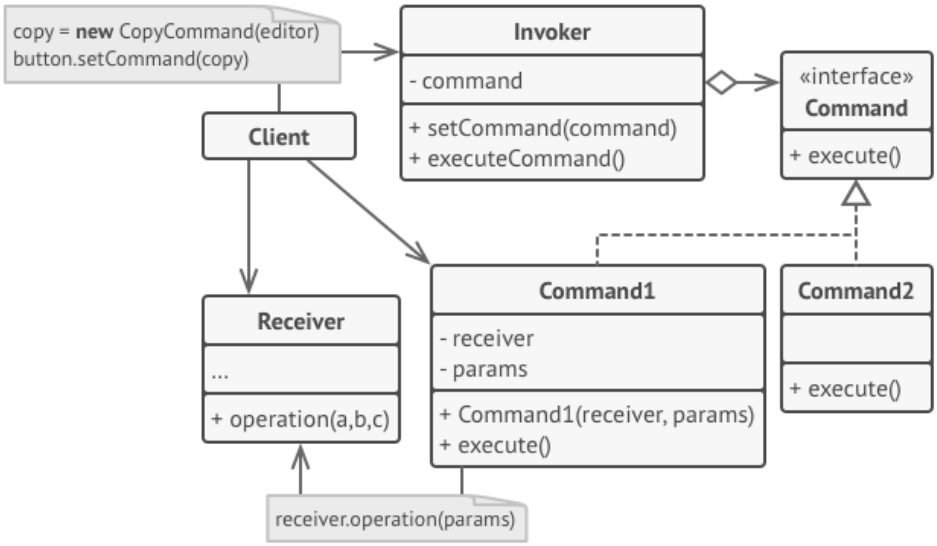


Рисунок 9.1.2 - Структура шаблону «команда»

Кожен екземпляр агенту може мати масив об’єктів-дій, унаслідуваних від даного інтерфейсу, наприклад: IMoveAction, IJumpAction, і.т.д. При цьому виникає проблема суміності агенту з RL-механізмом, на який він посилається, адже якщо простір дій кожного агенту можливо вільно задати, зникає гарантія того, що RL-модуль зможе працювати і різними агентами одного типу. Дану задачу можна вирішити, закріпивши визначення простору дій у конструкторі агенту, але це не вирішить основну проблему: для реалізації певної дії агентом у середовищі, як правило, необхідний доступ до його полів (наприклад для переміщення агенту у просторі потрібний доступ до його фізичної компоненти), які можуть бути захищені приватним модифікатором. Таким чином, обєкти наслідувані від інтерфейсу IAction

не можуть виконувати логіку агенту, якщо вони не є частиною коду агенту, але в такому випадку це нівелює необхідність використання інтерфейсу. Більш того, деякі дії для виконання можуть потребувати аргументів, які неможливо буде передати у загальний визначений метод Execute (окрім випадку якщо є певний клас-носій усіх можливих параметрів, що є громіздким негнучким рішенням). Таким чином, виникає проблема: як визначити простір можливих дій агенту у вигляді масиву окремих методів, які можуть потребувати різних аргументів та викликатися деяким зовнішнім RL-модулем, що не повинен мати даних про реалізацію дії, але повинен гарантовано знати скільки всього дій може виконати агент у будь-якому стані.

Для ініціалізації простору адаптивних дій можливо визначити окремий абстрактний метод, який повинен викликатися при створенні агенту генерувати словник з парами ключ-делегат, після чого повертати розмір словника дій для застосування конструктора RL-модуля, якщо він ще не був визначений. Таким чином, перший ініціалізований агент буде викликати початкову конфігурацію модулю прийняття рішень, який гарантовано задовільнить подальших агентів створених зі застосуванням того самого методу. Оскільки агенти не повинні визначати який RL-модуль відповідає їх типу (можуть існувати різні модулі прийняття рішень, що використовують різні RL-алгоритми і з точки зору агенту немає різниці як саме він працює), ця функція має виконуватися у класі-фабриці. Слід відмітити, що за даної моделі поняття «створення» та «ініціалізації» агенту становляться окремими функціями класу-генератору (перша відповідає за поміщення агенту у середовище і має бути абстрактною, друга – за визначення RL-модулю і має реалізуватися на рівні батьківського класу-фабрики), тому безпосереднє застосування шаблону «фабричний метод» стає неможливим. До того ж, необхідно визначити як реалізується зворотний зв'язок агенту з середовищем, адже після вибору певної дії, агент має оновити свій поточний стан і отримати певну винагороду згідно до поставленої задачі. Також, слід передбачити, що згідно до моделі навчання з підкріпленням, у шаблоні зобов’язані бути визначені наступні механізми:

– Агент повинен зберігати історію пар «стан - обрана дія» для вирахування кумулятивної винагороди

– Робота середовища повинна ділитися на певні ітерації у ході яких агенти отримуватимуть змогу виконати дію, результатом чого стане їх повідомлення про стан до якого вони перейшли та, опціонально, отримана винагорода.

– У випадку отриманні винагороди агент повинен надати історію рішень та винагороду відповідному до його типу RL-модулю для переоцінки Q-функції.

Розроблений шаблон, що відповідає даним умовам наведено на рисунку 9.3 та у додатку В.

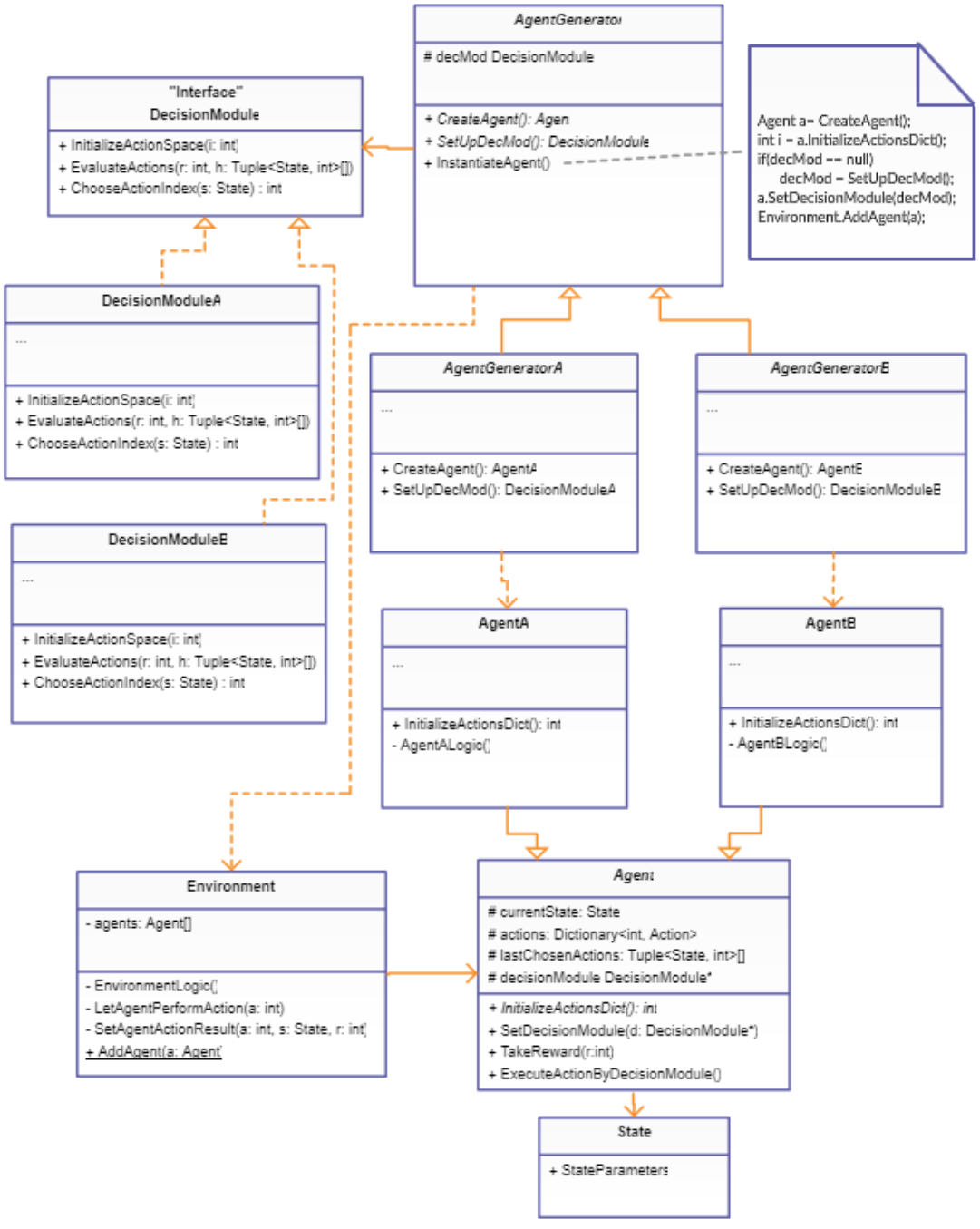


Рисунок 9.3 – Шаблон «Q-агент»

Як видно на схемі, шаблон є розширеною версією шаблону «фабричний метод», що містить весь необхідний функціонал для реалізації адаптивного механізму у ігровому середовищі. Алгоритм роботи даного шаблону наведено у додатку Г. Усі RL-агенти наслідуються від абстрактного класу Agent, який включає загальний для них функціонал, поля зі словником дій, історією дій та посилання на RL-модуль. Наслідувані екземпляри зобов'язані мати власну реалізацію InitializeActionsDict(), який визначає словник делегатів, що буде використовуватися для адаптивних впливів. Варто також відмітити, що застосування одного методу з різними параметрами можуть бути окремими значеннями у словнику. Стан є окремим класом, екземпляр якого є полем агенту, оскільки для кожної одиниці він свій і впливає на рішення що обираються. Стан можливо представити як набір параметрів, що використовується при виклику визначеного у базовому класі методу ExecuteActionByDecisionModule. Модулі прийняття рішень представлені класами, що реалізують інтерфейс DecisionModule, що включає такі функції, як: створення початкової Q-функції на основі переданої кількості допустимих дій, переоцінки реалізованої Q-функції на основі історії рішень агента та винагороди за них, і метод для вибору дії з множини дій на основі поточного стану шляхом отримання значення Q-функції. Клас AgentGenerator є абстрактним класом, що містить абстрактні методи: «CreateAgent», «SetUpDecMod», які обов'язкові до визначення у класах наслідниках (тут: AgentGeneratorA, AgentGeneratorB). Перший метод виконує ініціалізацію певного типу агенту в залежності від генератору, який викликає метод, другий – реалізує ініціалізацію необхідного модулю прийняття рішень в залежності від типу генератору. Метод InstantiateAgent є визначеним у базовому класі і відповідає за виклик перевантажених методів для створення нового агента, присвоєння значення полю decMod (якщо воно ще не має значення), надання агенту посилання на дане значення та його введення у середовище за допомогою статичного методу класу Environment (Можливо також реалізувати за допомогою шаблону «Одиночка»). Ігровий простір представлений окремим класом, хоча на практиці це може бути проміжний клас, що має доступ про параметри ігрового середовища і таким чином

може визначити стан агента. Функціональну схему даного шаблону наведено у додатку Д.

За наведеного шаблону процес навчання працює наступним чином:

1) Створений екземпляр агенту існує у ігровому середовищі, яке працює ітераціями, у ході яких, для усіх існуючих агентів викликається метод ExecuteActionByDecisionModule.

2) Агент звертається до екземпляру DecisionModule, на який він отримав посилання при ініціалізації та викликає публічний метод ChooseActionIndex, передавши свій поточний стан як аргумент.

3) Екземпляр DecisionModule обчислює значення Q-функції та повертає індекс дії згідно до обраної політики.

4) Агент виконує дію шляхом виклику делегату зі словнику actions за отриманим індексом та зберігає у список останніх рішень пару стан-індекс дії.

5) Даний процес продовжується доки дії агенту не призводять до певного результату з точки зору правил гри – позитивного чи негативного, в результаті чого середовищем для даного агенту викликається TakeReward з аргументом-винагородою.

6) TakeReward містить виклик EvaluateActions у екземплярі наявному DecisionModule з передачею отриманої нагороди та історії рішень у якості аргументів.

7) Об’єкт DecisionModule виконує перерахування Q-функції з градієнтним спуском до рішення, що дає вищу винагороду.

Діаграму послідовності наведено у додатку Е. Даний шаблон було використано для переписання програми дослідження ефективності агентів на основі дилеми в’язня. Діаграму класів наведено у додатку Ж. Як видно на діаграмі, застосування шаблону дозволило отримати гнучку систему, доступну для розширення новими ігровими механізмами та типами агентів. Також слід відмітити, що класи агентів були розвантажені для спрощення внутрішньої логіки.

9.2 Вхідні та вихідні параметри адаптивного механізму

Ігровий ШІ на основі навчання з підкріпленням дозволяє отримати оптимальну поведінку агенту за критерієм винагороди, але, технічно, вихідним може бути будь-який параметр. В залежності від цілей дизайнеру гри, будь-який ігровий механізм можна зробити адаптивним, визначивши простір рішень ШІ у вигляді маніпуляцій над змінними ігрового середовища, а винагороду ШІ – як значення, обернене до різниці між поточним та бажаним значенням таргетного адаптивного параметру. Тут, змінні якими маніпулює ШІ є вихідними параметрами, або перцепторами. Це – властивості ігрового середовища, зміна яких може призвести до змін значень рецепторів – вхідних параметрів, на основі значень яких формулюється винагорода, таким чином над ними виконується процес адаптації. Так, наприклад, зв’язавши через RL-систему функцію показник успішності гравця (згідно до правил гри) у вигляді вхідного параметру, та ваги у дереві прийняття рішень NPC можливо отримати систему регуляції складності, що підлаштовує свою стратегію для створення оптимального досвіду. Рецептори та перцептори залежать від жанру гри та особливостей її ігрових механік, їх вибір є задачею ігрового дизайнеру на основі його побажань щодо принципу роботі адаптивних механізмів. Можливі вхідні параметри наведені у таблицях 9.2.1 і 9.2.2, а вихідні у таблицях 9.2.3 та 9.2.4.

Таблиця 9.2.1 – Можливі вхідні параметри для RL-агенту

Показник навичок	Коментар
Програші на одиницю часу	Цей параметр може показати, наскільки важкою є гра для гравця. Співвідношення програшів до часу (може бути представлений як час у секундах, так і у ігрових одиницях, наприклад, ходів) у одиночних іграх, зазвичай, залишається приблизно на однаковому рівні протягом усього ігрового часу через одночасне зростання навичок гравця і складності гри. Постійне збільшення цього

Показник навичок	Коментар
	параметра може означати, що гра занадто важка для гравця, і складність може бути зменшена. І навпаки, постійне зменшення даного значення може означати небезпеку того, що гравцю стане нудно. Розробник може сам обрати порогові значення
Програші на рівень	Певні ігрові рівні можуть використовувати різні ігрові механіки. Різке зниження даного параметра порівняно з його середнім значенням означає, що гравець може мати проблеми з певними конкретними елементами гри, що використовуються на поточному рівні. Ця інформація може допомогти налаштувати ігрові механіки точніше, замість того, щоб зменшуючи повну складність і створювати незбалансований ігровий процес, що може роздратувати гравця.
Виграші/ Програші підряд	Залежно від жанру жанру співвідношення перемог до програшів (де поняття перемог та програшів визначається розробником і не обов'язково відповідає правилам гри) може краще підходити для адаптації ШІ. Наприклад, у покроковій стратегії гравець може мати необмежений час для ходу, тому важко визначити, чи виграє він або програє занадто часто або занадто рідко. У таких випадках краще відстежувати кількість послідовних однакових результатів гри.
Співвідношення причин програшів	Співвідношення причин програшів може допомогти визначити конкретний елемент гри, який турбує гравця. Це може бути певний різновид ворога, недостача деякого ресурсу, тощо. Цей параметр може використовуватися для точної корекції ігрових елементів.

Показник навичок	Коментар
K/D	Коефіцієнт вбивств/смертей (Kill/Death) є визначним параметром для більшості ігор в жанрі «боєвик». Максимізація цього параметру при якнайбільшій складності гри означає, що гравець досягнув оптимального ігрового досвіду.
Середній HP	Інформація про середнє значення здоров'я персонажа гравця (HP) може бути використана для налаштування параметрів, таких як ворожа точність, пошкодження, швидкість, і.т.д. Гравець відчуває себе більш напруженим, коли його HP вичерпується і сприймає перемогу як значніше досягнення, тому можливо прийняти міри, щоб статистично утримувати це значення в меншому інтервалі.
Час на рівень	Час, який гравець витратив на рівень, залежить від дизайну та розміру рівню, тому важко правильно оцінити цей параметр. Краще використовувати його у порівнянні із середньою статистикою всієї бази гравців.
Точність	Може бути використаною, в залежності від ігрового жанру, для оцінки навичок гравця

Таблиця 9.2.2 – Можливі вхідні параметри для RL-агенту

Показник підходу	Коментар
Застосування ресурсів	Більшість ігор пропонують різні типи ресурсів (аптечки, патрони, гроші, і.т.д.), які гравець може використовувати певним чином після їх знаходження. Порівняння частоти генерації ресурсів у ігровому світі з частотою використання їх використання дозволить

Показник підходу	Коментар
	виявити відносне надмірну експлуатацію певних ресурсів з боку гравця, або їх ігнорування. Якщо співвідношення не змінюється, це означає, що гравець занадто спирається на конкретну тактику, що може зробити ігровий процес монотонним і стати причиною нудьги. Такої ситуації можна уникнути, виштовхнувши гравця зі своєї зони комфорту, змінюючи частоту генерації корисних об'єктів, щоб надмірно експлуатовані предмети з'являлися рідше, що надасть їм додаткову цінність та змусить гравця більше імпровізувати.
Рівень агресії	Висока частота сутичок з NPC в RPG, агресивні автомобільні маневри в гоночних іграх, активне використання «фінішних приймоів» у файтингах і слэшерах, а також надмірне використання різних агресивних ігрових механік може дати точне уявлення про бажаний стиль гри гравця. Деякі гравці насолоджуються активним геймплеєм, коли інші намагаються звести до мінімуму кількість сутичок, щоб зберегти свої ресурси або просто тому, що їм не подобається брати участь у битвах. Слід враховувати, що у більшості ігор конфлікт – важлива частина, яку не слід відкидати лише тому що гравець намагається уникнути його. Ці дані можна використовувати для корекції ігрового середовища та елементів ігрової атмосфери.
Рівень напруги	Люди грають у ігри жахів, тому що їм подобається відчувати небезпеку, стрес і інші відчуття, що пов’язані з екстремальними умовами. Очевидно, що різні гравці можуть мати різні пороги страху, і деяких з них складно злякати, в той час як інші можуть вважати гру досить страшною. Іноді гравці можуть вибрати

Показник підходу	Коментар
	неправильний рівень складності на початку гри, тому що вони не знають, чого очікувати від неї, але згодом будуть відчувати дискомфорт, якщо зрозуміють що зробили неправильний вибір. Щоб перевірити, чи не стало гравцю нудно, AI може аналізувати такі параметри, як: тремтіння камери (гравці можуть виконувати надмірні дії, якщо вони відчувають страх), частота бігу (якщо гравець пробігає через невідоме місце перед зустріччю з ворогом, це може означати що він не відчуває себе в небезпеці), частота хования (якщо гравець намагається заховатися або присісти, коли бачить ворога, це означає, що він намагається бути обережним, протилежно до цього, ходьба поблизу ворогів означає, що гравець не розпізнає їх як реальну загрозу).
Моральні дилеми	Рішення гравця можуть надати дані про його психологічний портрет. Вибір у грі може бути непрямим (вибір, який гравець робить через використання ігрової механіки) і прямий (дилема з обмеженим числом варіантів, які гра надає гравцеві). Наприклад, геймдизайнер може використовувати систему «карми» для оцінки рішень гравця

Таблиця 9.2.3 – Можливі вихідні параметри RL-агенту

Фактор складності	Коментар
Параметри ворогів	Параметри ворожих персонажів мають прямий вплив на рівень складності гри, але зміна кожного з них може призвести до різного ефекту для різних гравців. Збільшення всіх з них одночасно може

Фактор складності	Коментар
	призвести до розриву між рівнями складності і роздратувати гравця, тому краще адаптувати ворожі параметри окремо, доки не буде створено оптимальну задачу для гравця. Ворожими параметрами можуть бути: рівень здоров'я (HP), броня, швидкість відповіді (реакція), обізнаність/увага, пошкодження, що наносяться гравцю, швидкість, точність, супротив різній зброї/інструментам/тактикам (це також можливо використати щоб запобігти надмірної експлуатації гравцем конкретної тактики та мотивувати його час від часу змінювати підхід).
Частоти появи ворогів	Для ігор у жанрі «виживання» добре підійде застсоування даного параметру у випадку, коли гравець вимушений витрачати цінні ресурси, коли зустрічає ворожих агентів. Частота появи ворогів має бути достатньою рівно настільки, щоб гравець постійно застосовував майже усі свої ресурси, що дозволить зберігати рівень напруженості. Але це може бути поганим підходом для воєнних шутерів або інших ігор, де кількість ворогів повинна бути константною для адекватного ігрового досвіду.
Частота генерування ресурсів	ШІ може керувати частотою появи у ігровому світі різних типів ресурсів, щоб мотивувати гравця постійно пристосовуватися і змінювати свою тактику гри. Також це може бути використано для надання гравцю додаткових інструментів, коли він найбільше потребує їх. Наприклад, ініціалізація додаткової аптечки у середовищі, коли гравець вичерпає медикаменти і має лише кілька HP, дозволить йому протягнути трохи довше і створить приємне відчуття «вдачі».

Фактор складності	Коментар
Приховані перки гравця	ШІ може активувати деякі приховані функції гри, коли немає можливості скоригувати параметри персонажів та ігрового середовища, щоб зробити гру більш або менш складною. Наприклад: змусити ворожих стрільців промахнутися при першому пострілі, надати критичний збиток останній кулі гравця в магазині, надати незначну допомогу у прицілюванні, надати додаткове значення балам здоров'я, коли вони майже вичерпаються і т.д.
Підказки	Ігри часто використовують систему підказок, щоб допомогти новому гравцеві навчитися грати. Деякі гравці часто можуть бути роздратовані, коли на екранах періодично з'являються непотрібні підказки, а в інших ситуаціях гравці можуть постраждати, оскільки вони їх не повідомили про цінні ігрові функції в правильний час. Іноді високий рівень складності автоматично вимикає підказки, але це неправильний підхід, тому що той факт, що гравець очікує на виклик, не означає, що він одразу повинен знати керування грою. Адаптивна механіка може стати корисною в цьому випадку. Аналіз навичок гравця дасть всі необхідні дані для усунення непотрібних підказок.
Тактика	Багато ігор мають градацію стратегій ШІ для різних рівнів складності. Наприклад, у шутерах NPC можуть використовувати комплексні бойові тактики на високих рівнях складності. Ця градація не завжди коректна, оскільки різні дії можуть бути по різному ефективними проти конкретних гравців. За допомогою адаптивної механіки розробник може підготувати набір тактик, які NPC могли б пробувати у процесі пошуку найбільш ефективних

Фактор складності	Коментар
	проти кожного гравця. Також, тактика може бути представлена у вигляді дерева рішень у якому порогові значення змінних для прийняття певних рішень будуть вихідними параметрами ШІ. Постійний пошук слабких місць у стилі гри гравця і використання їх буде мотивувати гравця більше думати про його дії і зробить виграш більш приємним.

Таблиця 9.2.4 – Можливі вихідні параметри RL-агенту

Фактор середовища	Коментар
Погодні умови та явища	Керування погодою та іншими властивостями середовища в деяких сценах на основі ігрового стилю гравця може створити сильну атмосферу і більше ефектних сцен.
Скриптові події	Якщо гра має набір заготовлених подій, які можуть випадково відбуватися під час гри, можна зробити залежність між стилем гри або моральним портретом гравця і сюжетними елементів, щоб зробити історію більш наближеною до гравця. Реалізація такої системи вимагала б класифікації всіх альтернативних сцен і методу оцінки, який будуть використовуватися для перевірки того, наскільки гравець задоволений історією.
Коригування дизайну середовища	Використання статистичні даних про ігровий процес для внесення змін у вигляд ігрового середовища може бути цікавою ігровою механікою, якщо правильно виконано. Наприклад, гра може

Фактор середовища	Коментар
	змінити колірну схему на більш темну, якщо гравець віддає перевагу агресивному стилю гри.
Поведінка NPC	Окрім дій що безпосередньо впливають на ігрове середовище, та дозволяють агентам досягнути їх мети, ігрові персонажі часто здатні виконувати інші форми поведінки, як, наприклад озвучувати репліки з голосового банку, змінювати idle-анимації та ін. Як і у випадку з дизайном середовища, NPC можуть змінювати характер своєї поведінки в залежності від обставин, наприклад, в залежності від напруженості подій. Даний параметр є неочевидним для реалізації з точки зору класичних методів розробки ШІ, але може бути продуктом існування адаптивної системи.

10 РОЗРОБКА СТАРТАП ПРОЕКТУ

В даному розділі проводиться опис стартап проекту.

10.1 Опис

Голова мета даної роботи – розробити методологію з імплементації навчання з підкріпленням у відеоігрових штучних інтелектах та перевірити її ефективність за допомогою серії тестів у ігровому середовищі. Узагальнення цієї ідеї можна побачити в таблиці 10.1.

Таблиця 10.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Застосування навчання з підкріпленням у відеоігровому штучному інтелекті	Імплементація не тренованих попередньо адаптивних механізмів	Реалізація у грі адаптивних механізмів, що здатні підлаштовуватися під зміну параметрів середовища, змінюючи принцип дії алгоритму ШІ по мірі необхідності.

Даний проект носить теоретичний характер, тому неможливо підняти питання про конкуренцію, але він може бути порівняний з сучасними широко використовуваними техніками реалізації ігрових ШІ, які можна визначити як «статичні». Окрім цього, для порівняння, будуть взяті ігрові ШІ на основі машинного навчання з попереднім тренуванням, які частіше використовуються у академічній сфері, але також можливі у прикладному застосуванні Порівняння наведено у таблиці 10.2

Таблиця 10.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

Техніко-економічні характеристики ідеї	(Потенційні) товари/концепції конкурентів			Слабка сторона	Нейтральна сторона	Сильна сторона
	Даний проект	ШІ на основі статичних алгоритмів	ШІ на основі машинного навчання з попереднім трен-ням			
Здатність розвиватися	Присутня	Відсутня	Присутня	-	-	+
Можливість точного налаштування поведінки	Присутня	Присутня	Відсутня	-	-	+
Надлюдський рівень ігрової стратегії	Відсутня	Відсутня	Присутня	-	+	-
Необхідність перенавчання у випадку змін ігрових параметрів	Відсутня	Відсутня	Присутня	-	-	+

10.2 Технологічний аудит ідеї проекту

Оскільки проект, з технічної точки зору, є шаблоном проектування, то він не потребує ніяких окремих технологій для використовування і може застосовуватися у будь-якому ігровому движку та мові програмування обраними розробником. Технології, у контексті яких використовується проект, можна побачити на таблиці 10.3.

Таблиця 10.3 – Технологічна здійсненність ідеї проекту

Аспект проекту	Технології її реалізації	Наявність технологій	Доступність технологій
Ігрове середовища	Ігровий движок	У відкритому доступі є, наприклад, такі: Unreal Engine, Unity	Вільні для використання
Адаптивний алгоритм	Мови загального програмування, що застосовуються у движках	У відкритому доступі є, наприклад, такі: C#, C++, JS	Вільні для використання

10.3 Аналіз ринкових можливостей стартап проекту

Попередня характеристика потенційного ринку стартап-проекту та характеристика потенційних клієнтів стартап-проекту представлені в таблиці X.4 таблиці X.5 відповідно. Проводиться визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 10.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	20-30
2	Загальний обсяг продаж, грн/ум.од	1500 грн/ум. од.
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Недискримінаційні якісні
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	70%

Таблиця 10.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Потреба створення адаптивних ігрових механік	1. Великий бізнес 2. Середній бізнес	Потребують реалізації механіки у різних ігрових середовищах	Точність роботи шаблону, можливість гнучкого налаштування ШІ

Фактори загроз та фактори можливостей представлені в таблиці 10.6 та в таблиці 10.7 відповідно.

Таблиця 10.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Крадіжка інтелектуальної власності	Крадіжка ідеї або ключової інтелектуальної інновації	Не є проблемою оскільки шаблон програмного проектування не можливо захистити правом інтелектуальної власності.
2.	Отримання несанкціонованого доступу сторонніми особами	Хакерська атака що може призвести до компроментації даних клієнтів	Залучення спеціалістів з інформаційної безпеки Використання засобів шифрування та резервного копіювання
3.	Відсутність ринку	Відсутність шляху збуту товару внаслідок помилкового орієнтування	Ретельний розгляд проблем потенційних клієнтів Консультації із спеціалістами Продовження роботи над шаблоном з метою вдосконалення
4.	Недостача капіталовкладень	Відсутні кошти до моменту виходу на ринок	Не потребує коштів для виходу на ринок
5	Програмні помилки або неефективні результати у тестовому проекті	У процесі розробки виникли програмні помилки внаслідок специфічної природи шаблону проектування	Затримка релізу проекту, продовження роботи над шаблоном з метою вдосконалення

Таблиця 10.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Отримання інвестицій	Отримання капіталу що необхідний для реалізації продукту	Не потребує капіталу для реалізації
2.	Успішна маркетингова політика	В результаті проведеної маркетингової політики отримана висока зацікавленість користувачів	Продовження роботи над вдосконаленням шаблону, написання статей, мануалів, створення тестових проектів у системах контрою версій у відкритому доступі
3.	Поглинання	Пропозиція купівлі розроблених технологій з метою її подальшого вдосконалення	Розвиток розроблених технологій у складі компанії-власника

Ступеневий аналіз конкуренції на ринку та аналіз конкуренції в галузі за М. Портером представлені в таблицях 10.8 та таблиці 10.9 відповідно. Конкурентний аналіз не спрямований на визначення можливостей, загроз і відшукування стратегічних невизначеностей, що можуть створюватися конкурентами, оскільки результати роботи знаходяться в відкритому доступі і питання конкуренції не постає.

Таблиця 10.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Олігополія	Відсутність конкурентів Велика ринкова сила Схожість використовуваних технологій	Інформування ринку щодо появи нового шаблону шляхом публікації статті
Внутрішньогалузева	Діяльність в одній галузі економіки Надання сервісів одного типу	Результати дослідження безкоштовні Збільшення кількості публікацій
Товарно-видова	Надання різних сервісів одного типу	Збільшення кількості публікацій
Цінова	Використання цін для покращення економічних умов збуту	Результати дослідження безкоштовні

Таблиця 10.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу		Висновки
Прямі конкуренти в галузі	Системи на основі статичних алгоритмів	Інтенсивна конкуренція, монополізація ринку збуту
Потенційні конкуренти	Розмір капіталовкладень,	Можливості входу на ринок забезпечить збільшення публікацій та

Складові аналізу		Висновки
	доступ до каналів розподілу, впізнаваність на ринку	демонстрація простоти використання технології за допомогою створення репозиторіїв з тестовими проектами. В результаті аналізу проектів на народно-громадських інтернет-платформах питання конкуренції виключене
Постачальники	Відсутні	Відсутні
Клієнти	Змінні витрати: виробничі непрямі дегресивні; -системи інформації: публікації, репозиторії у відкритому доступі -рівень чутливості до цін: споживачі орієнтовані на цінність продукту; -продуктова диференціація: якість; Методи контролю якості: тестування та профілювання	Клієнти диктують якість забезпечення сервісу, його доступність
Товари-замінники	Копіювання функціоналу, монополізація дистриб'юторів, демпінгування	Пропонування вигідних умов дистриб'юторам, вигідна цінова політика

Обґрунтування факторів конкурентоспроможності, порівняльний аналіз сильних та слабких сторін шаблону проектування та SWOT- аналіз стартап-проекту представлені в таблиці 10.10, таблиці 10.11 та таблиці 10.12 відповідно.

Таблиця 10.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Унікальність сервісу	Розроблений продукт представляє унікальну технологію імплементації адаптивних ігрових механізмів для проектів, де можливі зміни параметрів та особливостей ігрового середовища, що може призводити до необхідності комплексних оновлень статичних алгоритмів ШІ.
2.	Цінова політика	Отримання прибутку неможливе за рахунок відсутності проекту для продажу як такого

Таблиця 10.11 – Порівняльний аналіз сильних та слабких сторін шаблону проектування

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні зі статичними алгоритмами						
			–3	–2	–1	0	+1	+2	+3
1.	Унікальність сервісу	14						+	
2.	Цінова політика	13	+						

Таблиця 10.12 – SWOT- аналіз стартап-проекту

Сильні сторони: Якість та довготривалість Безкоштовність	Слабкі сторони: Відсутність прибутку
Можливості: Інвестиції Придбання дослідження компанією	Загрози: Застосування результатів дослідження без посилання на публікації

Альтернативи ринкового впровадження стартап-проекту представлені в таблиці 10.13.

Таблиця 10.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Розробка тестового демонстраційного проекту	Ймовірне	3 місяці
2.	Маркетингова кампанія для приваблювання користувачів	Неймовірна	1 місяць
3.	Пропонування безкоштовних тарифів	Неймовірна	1 місяць
Обрана альтернатива: Розробка власного демонстраційного проекту			

10.4 Розроблення ринкової стратегії проекту

Вибір цільових груп потенційних споживачів та визначення базової стратегії розвитку представлені в таблиці 10.14 та в таблиці 10.15 відповідно.

Таблиця 10.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Відеоігрові проекти з видавцем	Мала	30%	Відсутня	Низькі бар’єри входу
2.	Незалежні відеоігрові проекти	Висока	70%	Відсутня	Низькі бар’єри входу
3.	Академічне застосування	Середня	50%	Відсутня	Низькі бар’єри входу
Які цільові групи обрано: відеоігрові проекти з видавцем, незалежні відеоігрові проекти, академічне застосування					

Таблиця 10.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Надання результатів дослідження великому та	Вибірковий розподіл	Здатність протистояти прямим конкурентам	Стратегія диференціації

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
	середньому бізнесу		Низькі витрати Ефективна співпраця	

Визначення базової стратегії конкурентної поведінки та визначення стратегії позиціонування представлені в таблиці 10.16 та в таблиці 10.17 відповідно.

Таблиця 10.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні	Ні	Ні	Розширення первинного попиту

Таблиця 10.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Відповідність затвердженим характеристикам Висока ступінь надійності системи Безкоштовність	Стратегія диференціації	Формування регулярного попиту Збільшення разового використання послуги	Інноваційність технології Безкоштовність Простота використання

10.5 Розроблення маркетингової програми стартап проекту

Визначення ключових переваг концепції потенційного товару та опис трьох рівнів моделі товару представлені в таблиці 10.18 та в таблиці 10.19.

Таблиця 10.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Потреба в імплементації	Користувач може використати шаблон проектування для	Якість надання послуг Простота використання Цінова перевага

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
	адаптивних механізмів	створення ШІ, що здатні пристосовуватися до ігрового середовища за обраними параметрами	Зручність користування
2	Підвищення гнучкості відеоігрових ШІ	Підвищення гнучкості відеоігрових ШІ	Згідно до результатів тестів, технологія дозволяє підвищити гнучкість розроблюваних ШІ

Таблиця 10.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
І. Товар за задумом	Шаблон проектування, що дозволяє використовувати навчання з підкріпленням для імплементації ігрових адаптивних механізмів		
	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	Кількість		1 шт.
	Якість: стандарти якості написання технічної документації		
	Пакування: відсутнє		
	Марка: Відсутня		
	Тестовий програмний продукт		
	Програмний продукт, технічна підтримка та підписка на оновлення		
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності від копіювання відсутній			

Визначення меж встановлення ціни та формування системи збуту представлені в таблиці 10.20 та 10.21 відповідно.

Таблиця 10.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	0 грн	0 грн	0 – 12 000 000 грн./міс.	0 - 0 грн

Таблиця 10.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Закупівля не здійснюється	Інформування користувачів шляхом публікації статей	Канал одного рівня	Селективна з використанням комбінованого каналу збуту

Концепція маркетингових комунікацій представлена в таблиці 10.22.

Таблиця 10.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комуніка цій, якими користую ться цільові клієнти	Ключові позиції, обрані для позиціонуванн я	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Імплементація адаптивних механізмів у відеоігрових ШІ	Прямі офіційні	Послідовність в реалізації обраної позиції Доступність та об’єктивність інформації про дослідження Унікальність досліджень	Формування у цільової аудиторії обізнаності про появу нового шаблону проектування. Інформування користувачів про властивості та переваги продукту Інформування користувачів про нові способи використання відомого продукту,	Раціоналіст ична стратегія реклами у вигляді публікації статей

ВИСНОВКИ

В даній магістерській дисертації було виконано розроблено методологію імплементації засобів машинного навчання з підкріпленням у відеоігровому штучному інтелекті з метою отримання адаптивних механізмів прийняття рішень без попереднього тренування в умовах змінного середовища з контрольованими властивостями згідно до потреб розробника. Після аналізу існуючих аналогів у ігрових та академічних проектах було виявлено, що інтелектуальні системи прийняття рішень у реальних ігрових проектах застосовуються надзвичайно рідко через неможливість точного налаштування властивостей системи, яка виникає внаслідок процесу тренування алгоритму і тому існує у вигляді «чорного ящику». До того ж, у процесі розробки часто виникає потреба у зміні ключових ігрових механізмів, що призведе до необхідності перетренування ШІ і можливості виникнення несподіваної поведінки. У академічних проектах для тренування алгоритмів, зазвичай, використовують визначене ігрове середовище, а час тренування може обчислюватися годинами, так як головним критерієм є якнайвищий результат агента у грі, в той час як у ігрових проектах система не повинна приймати максимально ефективні рішення, так як це позбавить гравця можливості перемогти. Також аналіз показав, що одним з небагатьох вдалих методів застосування адаптивних механізмів є використання методики Q-навчання завдяки її принципу роботи заснованому на дослідженні середовища, спостереженні за винагородами і корекції політики прийняття рішень на основі отриманих даних, що є аналогом того, як в ігри грають реальні люди. Окрім того, даний метод не потребує навчальних даних, що робить можливим його застосування у реальному процесі розробки відеоігор, де немає можливості виконувати навчання зі вчителем, або воно можливе лише після збору статистики з великої групи гравців.

У ході дослідження було виконано серію експериментів з імплементацією адаптивних алгоритмів у ігровій системі на основі дилеми в’язня зі змінними правилами з урахуванням відсутності можливості попереднього навчання. Використання машинного навчання з підсиленням виявилося ефективним методом

управління поведінкою агентів за умов нестабільного ігрового середовища, давши 53% перемог у турнірах при 33% отриманих нейрогенетичним алгоритмом та 14% статичними алгоритмами (в тому числі оптимальним за початкових умов).

На основі отриманих результатів, аналізу існуючих проектів та правил ігрового дизайну було розроблено методику введення описаного механізму у реальних ігрових проектах. Методика включає в себе шаблон проектування на основі розширеного «фабричного методу», що наводить рішення таких проблем як: можливість створення RL-агентів, які здатні наслідувати ознаки і результати навчання, прив’язання типизованих кластерів агентів до окремих модулів прийняття рішень, організація простору дій агентів для їх індексування і оцінки Q-функцією, збереження можливості налаштовувати поведінку агентів за допомогою додаткових штучних політик прийняття рішень. Також, методологія включає поради до реалізації RL-агентів та перелік можливих адаптивних ігрових механізмів з вхідними та вихідними параметрами.

Використання описаних методів дозволить розробникам ігрових програм створювати ефективні системи прийняття рішень без попереднього тренування або побудови комплексних дерев рішень як для задання поведінки ігровим агентам, так і для адаптивного керування глобальними ігровими параметрами. На основі виконаних напрацювань був розроблений стартап проект.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lara-Cabrera, R. Game artificial intelligence: Challenges for the scientific community / R. Lara-Cabrera, M. Nogueira-Collazo. // CEUR Workshop Proceedings. – 2015. – №1394. – С. 1-12.

2. Rasmussen J. Are Behavior Trees a Thing of the Past? [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php

3. Yannakakis, Geogios N. Game AI revisited / N. Yannakakis, Geogios. // Proceedings of the 9th conference on Computing Frontiers. – 2012. – С. 285–292

4. Fullerton T. Game Design Workshop / T. Fullerton – Morgan Kaufmann, 2008. – С. 496

5. Rogers S. Level Up! The Guide to Great Video Game Design / S. Rogers. – John Wiley & Sons, 2010. – С. 520

6. Difficulty Levels And Why You Should Never Use Them [Електронний ресурс] – Режим доступу до ресурсу: http://www.roguesnail.com/difficulty-levels/.

7. Echo: Ex-Hitman devs bring machine learning to stealth games [Електронний ресурс] – Режим доступу до ресурсу: https://arstechnica.com/gaming/2017/08/echo-hitman-preview/

8. Black&White [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20010713040110/http://www.cgonline.com/previews/blackwhite-01-p1.html

9. Офіційна сторінка Forza Motosport 5 [Електронний ресурс] – Режим доступу до ресурсу: https://www.forzamotorsport.net/en-US/games/fm5.

10. Офіційний сайт Left 4 Dead [Електронний ресурс] – Режим доступу до ресурсу: http://www.l4d.com/game.html.

11. Giacomell E. DOOM Level Generation using Generative Adversarial Networks / E. Giacomello, P. Luca Lanzi, D. Loiacono // arXiv:1804.09154v1. – 2018. – С.4

12. Procedural Map Generation [Електронний ресурс] – Режим доступу до ресурсу: https://www.gridsagegames.com/blog/2014/06/procedural-map-generation/

13. Goodfellow I. J. Generative Adversarial Networks / I. J. Goodfellow, J. Pouget-Abadie, M. Mirzai, D. Warde-Farley // arXiv:1406.2661. – 2004. – С. 2-4

14. Tijs T.J.W. Dynamic Game Balancing by Recognizing Affect / T.J.W. Tijs, D. Brokken, W.A. IJsselsteijn.// Lecture Notes in Computer Science : book series – 2008. – № 5294. – С. 6.

15. Mnih V. Playing atari with deep reinforcement learning / V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, M. Riedmiller // arXiv:1312.5602. – 2013. – С. 4-8

16. Silver D. AlphaGo: Mastering the ancient game of Go with Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://research.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html

17. Funnel Algorithm [Електронний ресурс] – Режим доступу до ресурсу: http://ahamnett.blogspot.com/2012/10/funnel-algorithm.html

18. Sutton, R. S. Reinforcement Learning: An Introduction / R. S. Sutton, A. G. Barto // Cambridge, MA: MIT Press. – 1998. – С 37-41

19. Sutton, R. S. Reinforcement Learning: An Introduction / R. S. Sutton, A. G. Barto // Cambridge, MA: MIT Press. – 1998. – С 50-51

20. O'Donoghue B. The Uncertainty Bellman Equation and Exploration / B. O'Donoghue, I. Osband, R. Munos, V. Mnih // arXiv:1709.05380v4. – 2018.

21. Yuxi L. Deep Reinforcement Learning: an Overview / L. Yuxi // arXiv:1701.07274. – 2018.

22. Julian A. On "solving" Montezuma's Revenge [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@awjuliani/on-solving-montezumas-revenge-2146d83f0bc3

23. Liu R. The Effects of Memory Replay in Reinforcement Learning / R. Liu, J. Zou // Annual Allerton Conference on Communication. – №56 – 2018.

24. Lafferty J.D. Double Q-learning. Advances in Neural Information Processing Systems / J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor // NIPS. – №23. – 2010.

25. Горяшко А. П. Теория игр: от анализа к синтезу. Обзор результатов / А.П. Горяшко // Cloud of Science. – 2014. – С. 112-154.

26. Axelrod R. The Evolution of Cooperation / R. Axelrod // NY: Basic Books. – 1984. – С. 75.

27. Bagirov A. Introduction to Nonsmooth Optimization / A. Bagirov, N. Karmitsa // Springer International Publishing. – 2014. – С. 98-102

28. Optimization Problem Types. Nonsmooth Optimization. [Електронний ресурс] – Режим доступу до ресурсу: https://www.solver.com/nonsmooth-optimization

29. Karmitsa N. Nonsmooth Optimization (NSO). [Електронний ресурс] – Режим доступу до ресурсу: http://napsu.karmitsa.fi/nso/

30. Martin M. Using a Genetic Algorithm to Create Adaptive Enemy AI. [Електронний ресурс] – Режим доступу до ресурсу: https://www.gamasutra.com/blogs/MichaelMartin/20110830/90109/Using_a_Genetic_Algorithm_to_Create_Adaptive_Enemy_AI.php

31. Lara-Cabrera, R. Game artificial intelligence: Challenges for the scientific community / R. Lara-Cabrera, M. Nogueira-Collazo, C. Cotta, A. Fernández-Leiva // CEUR Workshop Proceedings. – 2016. – С. 65-70

32. Sutton R. S. Reinforcement Learning: An Introduction / R.S. Sutton, A. G. Barto // Cambridge, MA: MIT Press. – 1998. – С. 102-104

33. McLeod S. A. Skinner - operant conditioning. [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplypsychology.org/operant-conditioning.html

34. Rogers S. Level up! / S. Rogers // West Sussex: John Wiley & Sons. – 2014. – С. 14-15.

35. Csikszentmihalyi P. H. Flow: The Psychology of Optimal Experience / P.H. Csikszentmihalyi, M. Mirvis // The Academy of Management Review. – 1991. – 16(3). – С. 46-48.

36. Schell J. The Art of Game Design: A Deck of Lenses / J. Schell // Morgan Kaufmann, 2008. – С. 62-80.

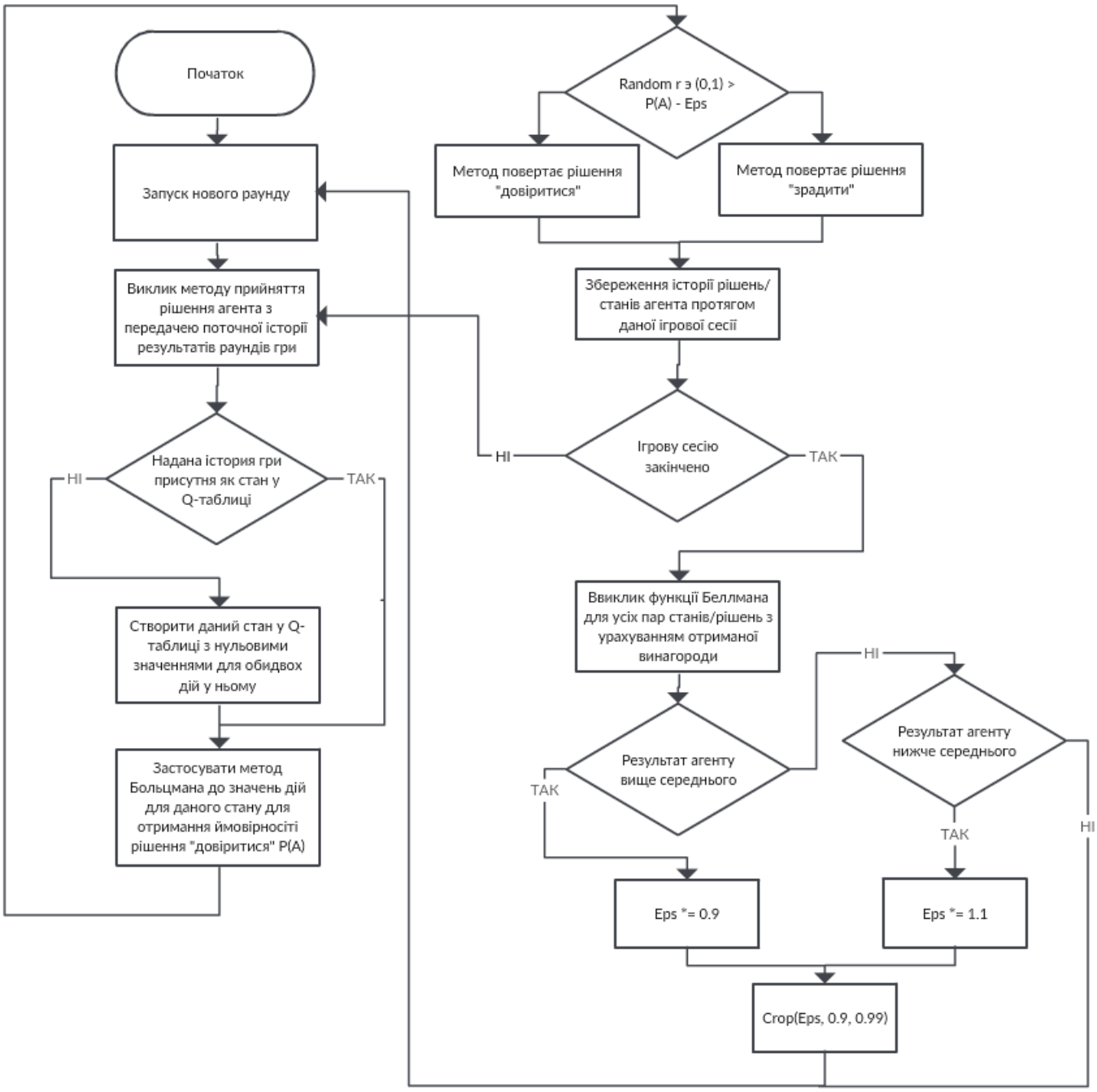
37. Helm R. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides // Addison-Wesley Professional; 1 edition. – 1994. – С. 12-15.

38. Паттерн проектування «Фабричний метод» [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/factory-method

39. Паттер проектування «Команда» [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/ru/design-patterns/command

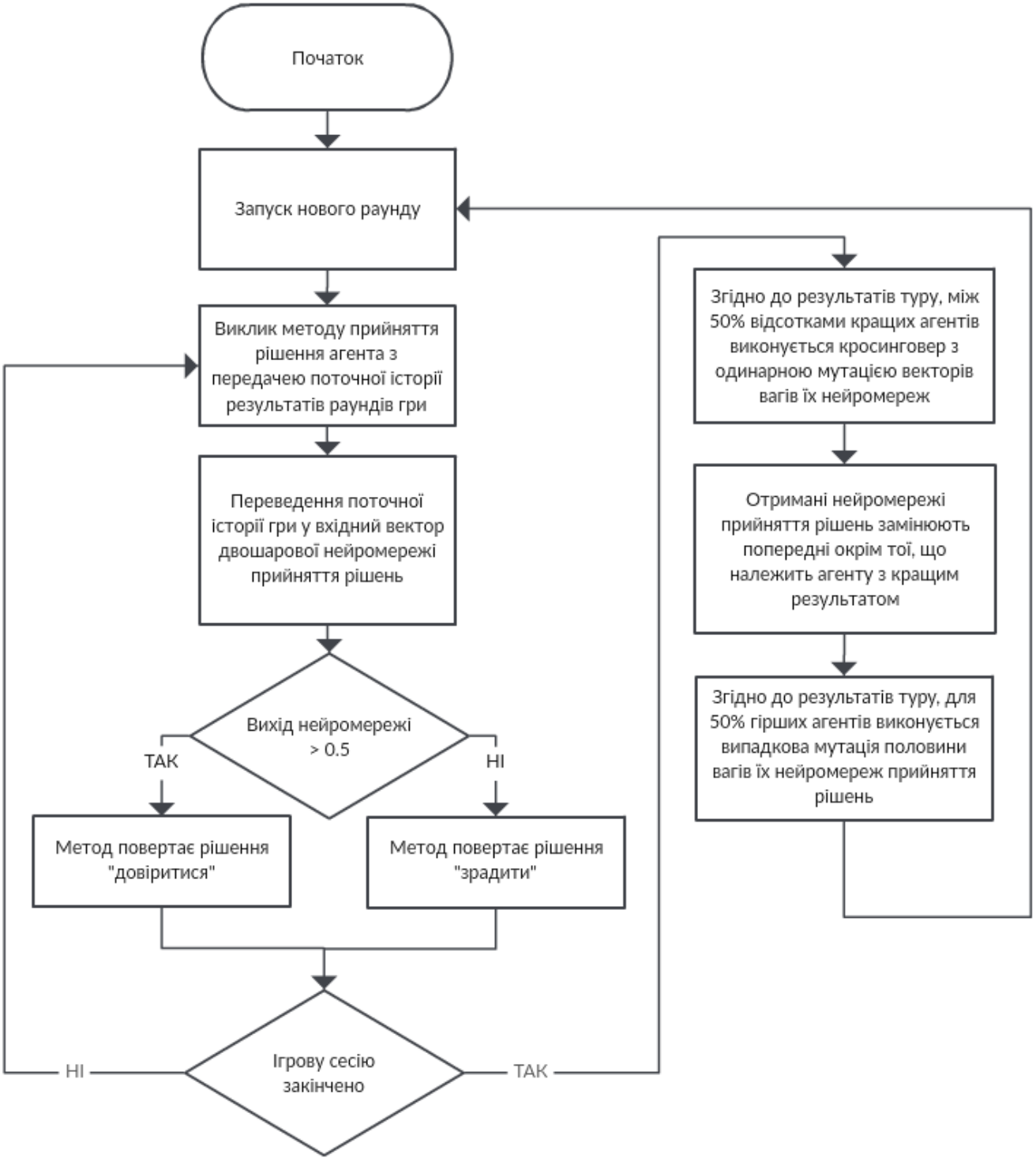
ДОДАТОК А

Алгоритм роботи RL-агента



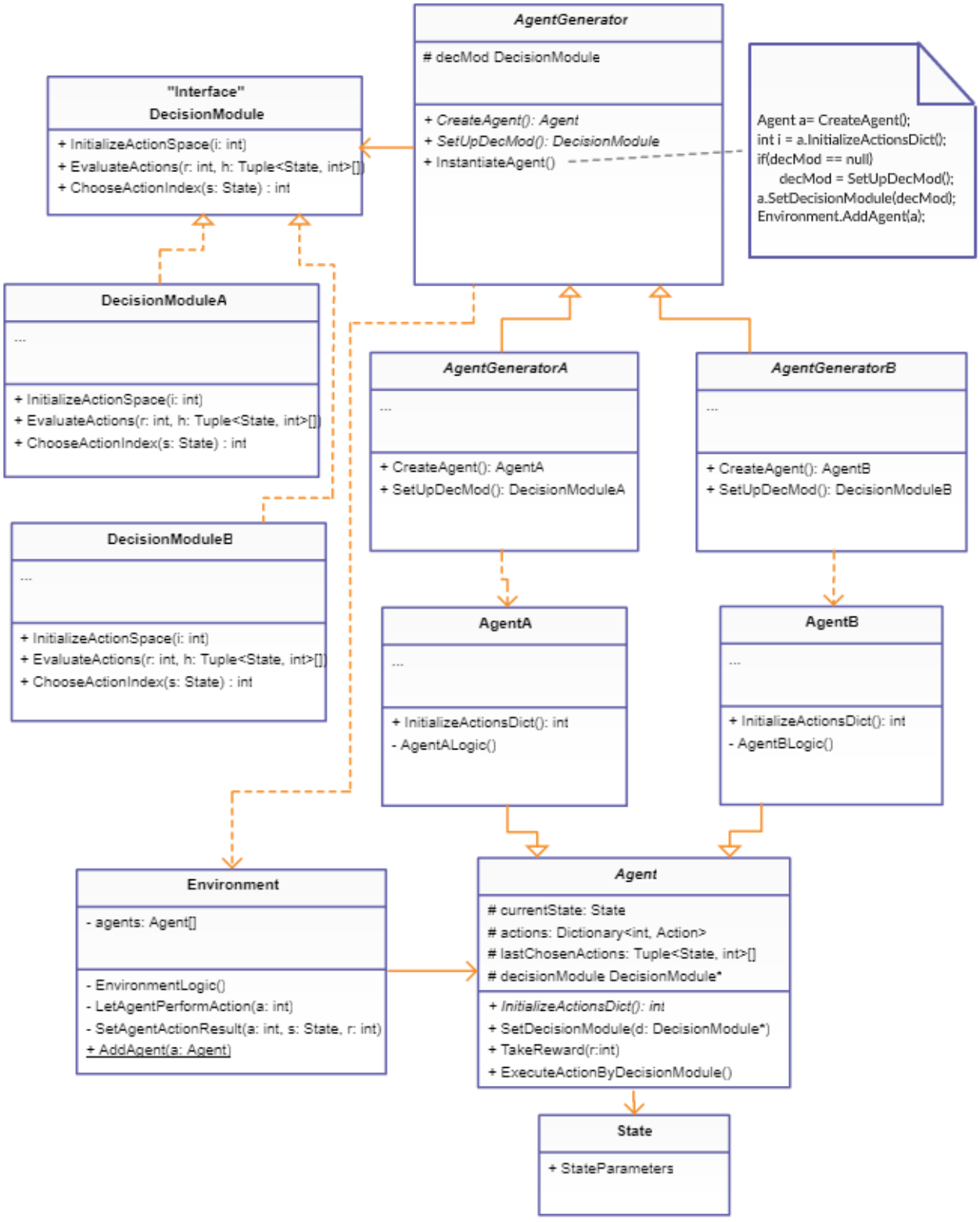
ДОДАТОК Б

Алгоритм роботи генетичного агента



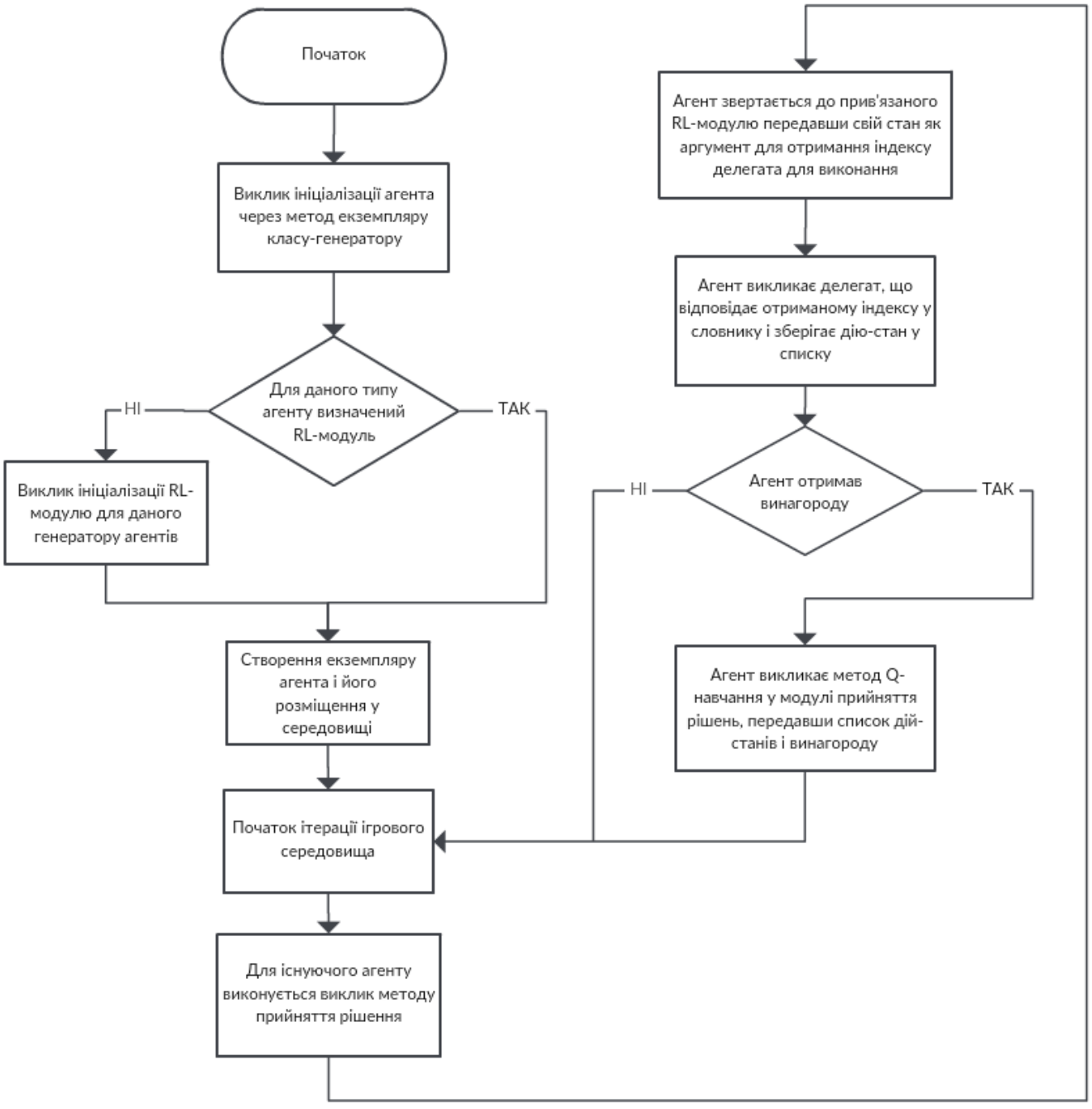
ДОДАТОК В

Шаблон проектування Q-агент



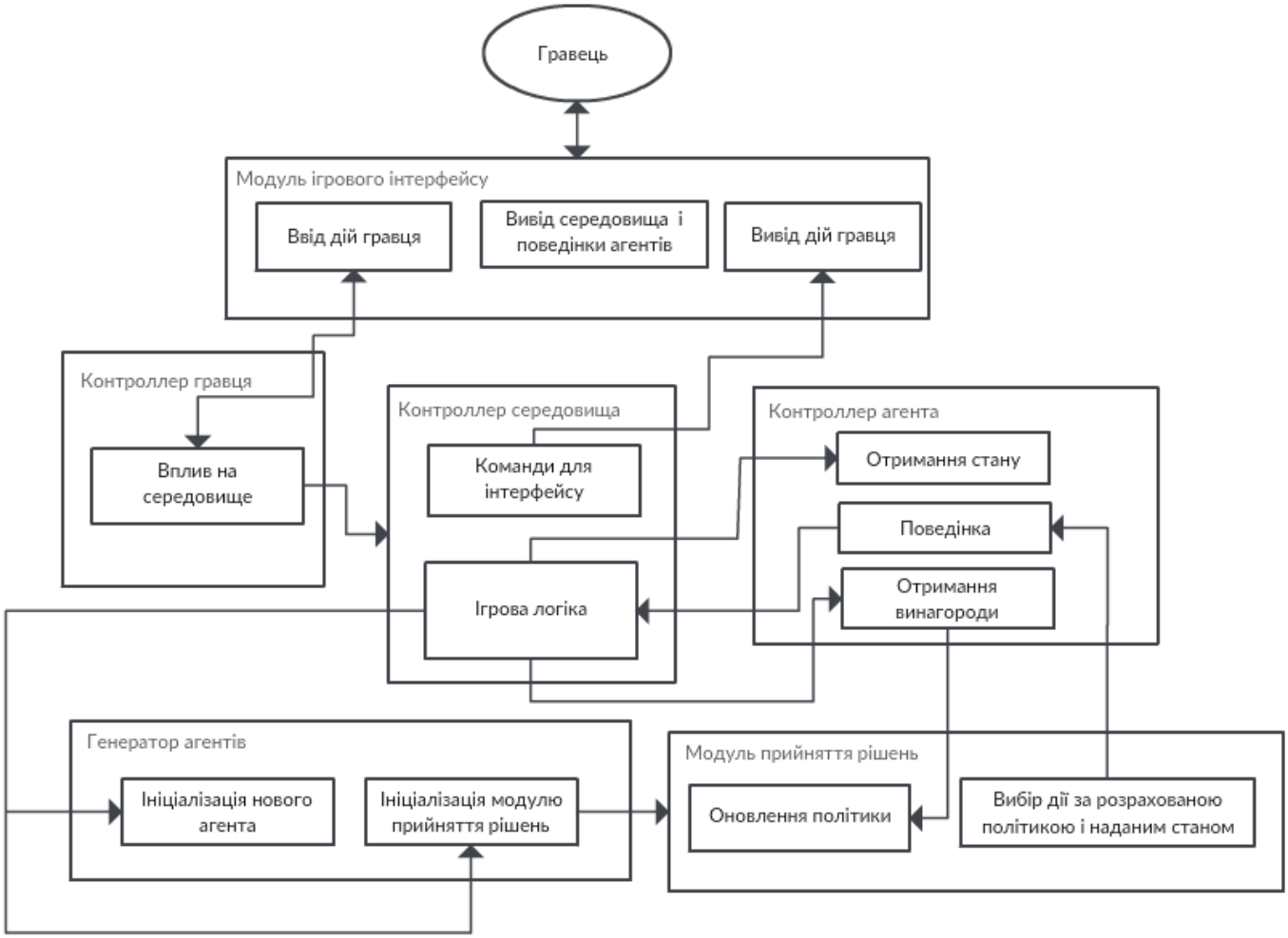
ДОДАТОК Г

Алгоритм роботи шаблону Q-агент



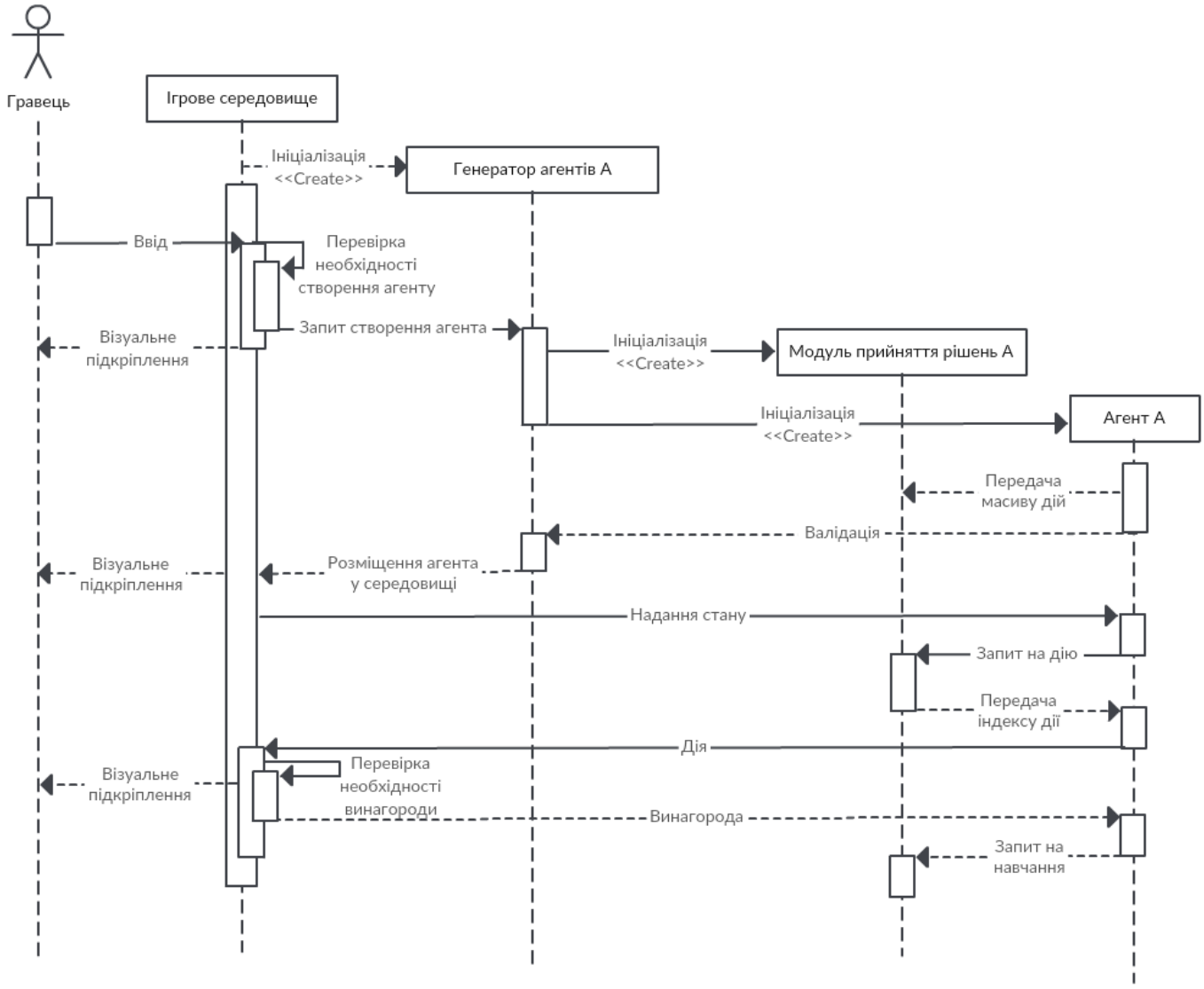
ДОДАТОК Д

Функціональна схема системи з застосуванням Q-агента



ДОДАТОК Е

Діаграма послідовності



ДОДАТОК Ж

Застосування Q-агента для дилеми в’язня